Contributions à la détection des comportements malhonnêtes dans les réseaux ad hoc AODV par analyse de la confiance implicite

Mohamed Ali Ayachi

N° d'ordre : 4223 ANNÉE 2011

THÈSE DE DOCTORAT

Université de Rennes 1 / Université 7 Novembre à Carthage
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention : Informatique
École doctorale : Matisse

et

DOCTEUR DE L'ÉCOLE SUPÉRIEURE DES COMMUNICATIONS DE TUNIS
Mention : Technologie de l'information et de la communication

présenté par

Mohamed Ali Ayachi

Contributions à la détection des comportements malhonnêtes dans les réseaux ad hoc AODV par analyse de la confiance implicite.

Thèse à soutenir à Rennes le 24/02/2011

devant le jury composé de :

Isabelle CHRISMENT
Professeur, Université Henri Poincaré / rapporteur

Belhassen ZOUARI
Professeur, Faculté des Sciences de Tunis / rapporteur

Sihem GUEMARA
Professeur, Sup'Com Tunis / examinatrice

Bernard COUSIN
Professeur, IRISA / examinateur

Adel BOUHOULA
Professeur, Sup'Com Tunis / directeur de thèse

Christophe BIDAN
Professeur, Supélec Rennes / directeur de thèse

Nicolas PRIGENT
Enseignant-Chercheur, Supélec Rennes / co-directeur de thèse

TABLE DES MATIÈRES

Table des matières **i**

Table des figures **vi**

Liste des tableaux **vii**

Introduction **1**

1 État de l'art **5**
1.1 Concepts fondamentaux des protocoles de routage ad hoc 6
1.1.1 Protocoles de routage ad hoc pro-actifs 6
1.1.1.1 Destination Sequence Distance Vector (DSDV) 7
1.1.1.2 Optimized Link State Routing (OLSR) 9
1.1.2 Protocoles de routage ad hoc réactifs 10
1.1.2.1 Dynamic Source Routing (DSR) 11
1.1.2.2 Ad hoc On-demand Distance Vector (AODV) 12
1.1.3 Protocoles de routage ad hoc hybrides 14
1.2 Sécurité du routage dans les réseaux ad hoc 17
1.3 Solutions et mécanismes de sécurité 22
1.3.1 Solutions basées sur la cryptographie 22
1.3.2 Systèmes de gestion de la réputation 25
1.3.3 Systèmes de gestion de la confiance 26
1.4 Positionnement 28

2 Analyse de la confiance **31**
2.1 Vulnérabilités du protocole de routage ad hoc AODV 32
2.1.1 Attaques élémentaires portant sur les demandes de route 34
2.1.1.1 Suppression d'une demande de route 34
2.1.1.2 Modification d'une demande de route 34
2.1.1.3 Fabrication d'une demande de route 35
2.1.1.4 ***Rushing*** d'une demande de route 35

2.1.2 Attaques élémentaires portant sur les réponses de route 35
2.1.2.1 Suppression d'une réponse de route. 35
2.1.2.2 Modification d'une réponse de route. 35
2.1.2.3 Fabrication d'une réponse de route. 36
2.1.3 Attaques élémentaires portant sur les erreurs de route 36
2.1.3.1 Suppression d'une erreur de route 36
2.1.3.2 Modification d'une erreur de route 36
2.1.3.3 Fabrication d'une erreur de route 37
2.1.4 Attaques composées 37
2.1.4.1 Répétition régulière d'attaques élémentaires 37
2.1.4.2 Insertion dans une route déjà établie 37
2.1.4.3 Insertion dans une route non encore établie 38
2.1.4.4 Création d'une boucle de routage 39
2.1.4.5 Création d'un tunnel 39
2.2 Analyse de la confiance implicite dans AODV 40
2.2.1 Présentation du langage de formalisation 40
2.2.2 Notations 40
2.2.3 Processus de découverte de route 42
2.2.3.1 Initialisation de la demande de route 42
2.2.3.2 Propagation de la demande de route 43
2.2.3.3 Propagation de la réponse de route 48
2.2.4 Maintient des routes 51
2.3 Détection des nœuds malhonnêtes 52
2.3.1 Présentation de la table d'historique étendue THE 53
2.3.2 Critères d'incohérences 54
2.3.2.1 Rejeu de messages 54
2.3.2.2 Fabrication de messages 56
2.3.2.3 Modification de messages 57
2.4 Résumé 59

3 Expérimentations **61**
3.1 Environnement de simulation 62
3.2 Choix de l'implémentation d'AODV 63
3.3 Implémentation des critères d'incohérences 65
3.4 Implémentation du comportement malhonnête 67
3.5 Mise en place des simulations dans NS-2 70
3.6 Résultats des simulations 71
3.6.1 Performance du système de détection 71
3.6.1.1 Résultats concernant les attaques élémentaires. 71
3.6.1.2 Résultats concernant les attaques complexes. 74
3.6.1.3 Discussion concernant les faux-positifs. 76
3.6.2 Performance du protocole 77
3.6.2.1 Taux de paquets reçus avec succès 78
3.6.2.2 Délai de bout-en-bout 79
3.6.2.3 Charge de routage 80
3.6.3 Mise à l'épreuve du système de détection 83
3.7 Résumé 85

4 Étude des faux-positifs **87**
4.1 Approche 1 . . . 87
4.1.1 Modifications apportées au critère d'incohérence . . . 88
4.1.2 Implémentation . . . 90
4.1.3 Résultats expérimentaux . . . 91
4.2 Approche 2 : complément de détection . . . 94
4.2.1 Principe . . . 94
4.2.2 Mise en place . . . 94
4.2.2.1 Cas 1 : décider puis traiter . . . 95
4.2.2.2 Cas 2 : traiter puis décider . . . 96
4.2.3 Résultats expérimentaux . . . 97
4.3 Mesures prises à l'encontre des nœuds malhonnêtes . . . 100
4.3.1 Types de mesures . . . 100
4.3.1.1 Exclusion temporaire . . . 100
4.3.1.2 Exclusion définitive . . . 101
4.3.2 Comparaison de l'impact de chaque mesure . . . 101
4.4 Résumé . . . 102

Conclusion **105**

A Rappel des notions cryptographiques **109**
A.1 Cryptographie symétrique . . . 110
A.2 Cryptographie asymétrique . . . 110
A.3 Fonctions de hachage cryptographiques . . . 111
A.4 Message Authentication Code . . . 112
A.5 Signature numérique . . . 112
A.6 Certificat électronique . . . 113

B AODV **115**
B.1 Structures maintenues par chaque nœud . . . 115
B.1.1 Table de routage . . . 115
B.1.2 Table d'historique . . . 116
B.2 Structures des messages échangés . . . 116
B.2.1 Demande de route RREQ (*Route REQuest*) . . . 116
B.2.2 Réponse de route RREP (*Route REPly*) . . . 117
B.2.3 HELLO . . . 118
B.2.4 Erreur de route RERR (*Route ERRor*) . . . 118
B.2.5 Accusé de réception de réponse de route RREP-ACK . . . 119

C Pseudo-code de l'implémentation AODV-UU **121**

Bibliographie **127**

TABLE DES FIGURES

1.1 Mise à jour incrémentale 7
1.2 Mise à jour complète (*full dump*) 8
1.3 Avantage de l'utilisation des MPR 9
1.4 Choix des relais multi-point pour le nœud 1 10
1.5 Exemple du processus d'établissement de route entre 1 et 5 12
1.6 Exemple d'établissement de route entre 1 et 5 14
1.7 Zone de routage de rayon=2 du nœud 1 et 4 15
1.8 Topologie dans un réseau implémentant ZHLS 16
1.9 Attaque trou de ver 20
1.10 Attaque par détour 20
1.11 Principales pistes de recherche 22
1.12 Authentification lors de la découverte de route à chaque saut dans ARAN 24

2.1 Invasion de route (route déjà existante) 38
2.2 Invasion de route (lors de l'établissement du chemin) 38
2.3 Création d'une boucle de routage dans une route déjà existante 39
2.4 Attaque du tunnel 39
2.5 Initialisation de la demande de route par S 43
2.6 Retransmission des RREQ par les nœuds intermédiaires 47
2.7 Propagation de la réponse de route RREP 50

3.1 Représentation schématique de NS-2 63
3.2 Imbrication des appels de méthodes dans AODV-UU 64
3.3 Ajout de la fonction `tap()` 64
3.4 Impact de la variation de la densité du réseau (trafic et nombre d'attaquants constants) 72
3.5 Impact de la variation du trafic dans un réseau de 40 nœuds 73
3.6 Impact de la variation du trafic dans un réseau de 100 nœuds 73
3.7 Impact de la variation de la densité du réseau (attaques complexes) 74
3.8 Impact de la variation de la densité du trafic (attaques complexes) 75
3.9 Attaques complexes vs attaques élémentaires 75
3.10 Taux de faux-positifs pour l'attaque par modification du `#SNS` dans une RREQ . . . 76

3.11 Taux de paquets émis/reçus avec succès (PDR) 78
3.12 Délai de bout-en-bout 79
3.13 Proportion des paquet de routage par rapport aux paquets de données (TOH) . . . 80
3.14 Charge de routage normalisée (NRL) 81
3.15 Taux de paquets délivrés avec succès pour l'attaque sur les paquets de données . . . 82
3.16 Taux de paquets délivrés avec succès pour certains couples émetteur/récepteur . . 82
3.17 Impact de la variation du nombre d'attaquants pour un réseau de 40 nœuds 83
3.18 Impact de la variation du nombre d'attaquants pour un réseau de 100 nœuds . . . 84

4.1 Impact de la variation de la densité du réseau 91
4.2 Impact de la variation de la densité du trafic 92
4.3 Impact de la variation du nombre d'attaquants dans un réseau de 40 nœuds 93
4.4 Variation de la densité du trafic dans un réseau de 40 nœuds 93
4.5 Impact de la variation de la densité du réseau en utilisant le complément de détection 98
4.6 Influence de la variation de la densité du réseau sur le PDR 98
4.7 Impact de la variation de la densité du trafic dans un réseau de 40 nœuds en utilisant le complément de détection 99
4.8 Impact de la variation de la densité du trafic dans un réseau de 100 nœuds en utilisant le complément de détection 99
4.9 Impact de la variation de la densité du réseau 101
4.10 Impact de la variation de la densité du trafic dans un réseau de 40 nœuds 101

A.1 Chiffrement symétrique 110
A.2 Chiffrement asymétrique 111
A.3 Utilisation des fonctions de hachage cryptographique 112
A.4 Utilisation des fonctions de hachage à clé symétrique 112
A.5 Signature numérique 113

B.1 Format de la table de routage 115
B.2 Format de la table d'historique 116
B.3 Format de la demande de route RREQ 116
B.4 Format de la réponse de route RREP 117
B.5 Format du message HELLO 118
B.6 Format de l'erreur de route RRER 118
B.7 Format de l'accusé de réponse de route RREP-ACK 119

LISTE DES TABLEAUX

1.1 Récapitulatif des attaques présentées 21
1.2 SEAD : Calcul des chaines de haché pour assurer l'authentification dans un réseau de diamètre $m = 5$, i est le numéro de séquence, j est le nombre de saut et la longueur de la chaine de haché est $n = 20$ 23

2.1 Récapitulatif des attaques élémentaires présentés dans [NS03] 33
2.2 Taxonomie des événements élémentaires anormaux [HL04] 33
2.3 Modifications possibles sur les champs des RREQ 34
2.4 Modifications possibles sur les champs des RREP 36
2.5 Modifications possibles sur les champs des RERR 37
2.6 Changements des tables de routage après retransmission de la RREQ 47
2.7 Changements des tables de routage après retransmission de la RREP 51

3.1 Paramètres de simulation dans NS-2 70

INTRODUCTION

Motivations et problématique

La constante évolution des technologies de l'information et le penchant vers l'utilisation des machines sans fil qui se sont imposées ces dernières années ont fait émerger un nouveau type de réseaux : les réseaux sans-fil ad hoc, ou MANET (*Mobile Ad hoc NETwork*). Se souciant de pouvoir communiquer et de partager l'information dans n'importe quelle situation, les réseaux ad hoc sont des systèmes autonomes composés par un ensemble d'entités mobiles libres de se déplacer sans contraintes. Ces entités utilisent le médium radio pour communiquer et forment un réseau n'utilisant aucune infrastructure existante. De ces faits, ces réseaux qualifiés de spontanés présentent une architecture originale qui évolue à tout instant.

Il existe plusieurs domaines d'application aux réseaux ad hoc. Le domaine militaire et celui des secours en cas de catastrophes restent des exemples fréquemment cités. Toutefois, plusieurs autres applications des réseaux ad hoc ont vu le jour. Nous citons les réseaux véhiculaires résultant de l'interconnexion de véhicules en mouvement ou les réseaux de capteurs capable de récolter et de transmettre les données environnementales. D'autres situations de la vie courante sont adaptées à l'utilisation des réseaux ad hoc. C'est le cas par exemple du réseau créé entre un professeur et ses étudiants pour le besoin d'une séance de cours ou le réseau créé entre les participants à une réunion ou même entre les voyageurs dans un train.

Le routage est une fonction primordiale dans les MANET où chaque entité mobile joue le rôle d'un routeur et participe activement dans la transmission des paquets de données. Ainsi, une entité peut communiquer directement avec une autre si elle est dans sa portée radio. Sinon, elle compte sur la coopération des voisins pour relayer ses messages. Cette équivalence entre les entités fait que les schémas classiques de routage utilisés dans les réseaux filaires ne s'appliquent plus pour les réseaux ad hoc, qui nécessitent donc la mise en place de protocoles de routage spécifiques. Ces

protocoles de routage ad hoc spécifient la manière avec laquelle les entités communiquent pour échanger des informations sur la topologie leur permettant de construire leur propre vision du réseau. Il existe trois catégories de protocoles de routage ad hoc, pro-actifs, réactifs et hybrides qui se différencient par le mode de fonctionnement de la phase de découverte du chemin et de la mise à jour d'informations de routage.

Les protocoles de routage ad hoc tel que conçus manquent de contrôles de sécurité. La plupart font l'hypothèse d'un comportement honnête entre les entités qui collaborent. La réalité peut toutefois être très différente en présence d'entités malveillantes capables de corrompre le bon déroulement des opérations pour servir leur intérêt.

Chaque entité doit mettre en œuvre ses propres mécanismes de sécurité pour se protéger des entités malhonnêtes, c'est à dire des entités en qui elle pourrait *a priori* avoir confiance mais qui ne sont, dans les faits, pas dignes de celle-ci.

De nombreuses propositions ont été faites en vue d'assurer la sécurité des protocoles de routage dans les réseaux ad hoc. Ces solutions peuvent être classées en 3 catégories :

- Les solutions basées sur la cryptographie font la distinction entre les entités qui sont autorisées à prendre part au réseau (et qui sont supposées se comporter correctement) et les entités qui n'y sont pas autorisées et qui sont considérées comme étant des attaquants. Ces mécanismes souvent lourd à mettre en place, n'empêchent pas les entités ayant le matériel cryptographique nécessaire de se comporter malhonnêtement.
- Les solutions à base de systèmes de réputation font calculer par chaque nœud une opinion sur les autres entités. Cette opinion est une valeur numérique résultant de la combinaison d'observation directes ou indirectes selon le cas. Les systèmes de réputation sont généralement utilisés pour détecter les entités égoïstes et les forcer à coopérer. Cependant, une entité malhonnête peut avoir une bonne réputation puisqu'elle participe aux opérations du réseau, tout en se comportant mal car elle diffuse de fausses informations.
- Les solutions à base de systèmes de gestion de confiance usent des propriétés intrinsèques des protocoles de routage pour chercher des incohérences entre les messages reçus. Ces solutions opèrent comme des systèmes de détection d'intrusions contre des attaquants internes.

Ce dernier type de solutions nous intéresse plus particulièrement puisque la notion de confiance est omni-présente dans les protocoles de routage ad hoc dans lesquels les nœuds se font mutuellement confiance et se basent sur la coopération des entités.

Objectif et contributions

Dans cette thèse, nous nous intéressons à sécuriser les protocoles de routage ad hoc en proposant des mécanismes de détection des actions malveillantes pour consolider les protocoles de routage et prévenir les futures attaques. Nous nous basons sur la confiance développée entre les entités pour bâtir un raisonnement sur le comportement des entités voisines. Ceci passe par une comparaison des messages échangés entre les entités du réseau ad hoc ou encore par l'analyse des incohérences dans les annonces et les ouvertures de routes afin de détecter la malhonnêteté ou la compromission d'une entité mobile.

Nous choisissons d'appliquer cette approche sur un protocole réactif (AODV [PBRD03]) étant

donnée que la majorité des solutions à base de systèmes de gestion de confiance ont été effectuées sur le protocole pro-actif OLSR [CJ03]. Nos contributions se présentent comme suit :

1. Analyser et formaliser la confiance implicite qui existe entre les entités dans un réseau exécutant le protocole de routage ad hoc AODV. Cette analyse concerne le comportement normal d'une entité qui, selon la phase (établissement ou maintient de route), est disséqué en actions élémentaires. Les règles obtenues concernent la confiance développée par une entité envers une autre et qui, selon les circonstances, exécute des actions en conséquence.

2. Dégager les critères qui permettent de déceler les incohérences. Les dits critères permettent de contrôler le comportement des voisins et de détecter toute déviation par rapport au comportement normal. Ils sont présentés sous forme de critères d'incohérences, des règles correspondant à la méfiance que les entités développent envers les entités malhonnêtes.

3. Implémenter et évaluer le système de détection. Les résultats obtenus prouvent la pertinence du raisonnement sans pour autant influencer les performances du protocole. Les limites sont aussi évaluées.

4. Étude des cas de faux positifs et proposition de solution pour les éliminer. Les expérimentations que nous avons effectués montrent des cas de faux positifs. Nous cherchons l'origine de ces fausses accusations et nous proposons une solution que nous testons et validons.

Organisation du mémoire

Ce mémoire est organisé de la manière suivante :

Dans le chapitre 1, nous présentons les concept fondamentaux des protocoles de routage ad hoc notamment à travers la présentation d'exemples de protocoles ainsi que les menaces de sécurité auxquels ils font face. Ensuite, nous passons en revue l'état de l'art des mécanismes utilisés pour assurer la sécurité. Nous terminons par nous positionner par rapport à ces travaux et argumenter nos choix.

Dans le chapitre 2, nous nous intéressons particulièrement au protocole de routage ad hoc AODV. Nous étudions les vulnérabilités sur lesquelles des entités malhonnêtes se basent pour mettre en place des attaques. Nous analysons ensuite la confiance implicite dans AODV, qui est représentée sous forme de règles mettant en valeur les actions élémentaires faites par chaque entité lorsque certaines conditions sont réunies. Nous présentons par la suite un ensemble de critères d'incohérences permettant la détection d'actions malhonnêtes.

Le chapitre 3 présente les expérimentations du système de détection introduit dans le chapitre précédent. Après avoir défini l'environnement de simulation ainsi que l'implémentation faite, nous étudions les performances du système de détection pour prouver son efficacité. Ensuite, nous vérifions que cette extension ne dégrade pas les performances du protocole. Enfin, nous testons les limites de notre proposition.

Le dernier chapitre (4) analyse les cas de faux positifs et propose une amélioration pour les supprimer. Nous présentons l'implémentation de cette solution ainsi que les résultats obtenus après simulations. Nous terminons par une discussions concernant les mesures prises à l'encontre des nœuds malhonnêtes.

Nous concluons ce document en faisant le bilan de nos contributions et en proposant des perspectives de travaux futurs.

CHAPITRE 1

ÉTAT DE L'ART

Les réseaux sans-fil ad hoc ou MANET (*Mobile Ad hoc NETwork*) sont des systèmes autonomes composés par un ensemble d'entités mobiles appelées usuellement nœuds qui communiquent au moyen du médium radio. Outre la mobilité, ces nœuds ont des ressources limitées et sont équivalents dans la fonction de routage. En ce sens, ces réseaux se distinguent par un déploiement facile, rapide et robuste car ils ne requièrent pas d'infrastructure pré-existante et d'autorité centralisée. Ils sont aujourd'hui essentiellement déployés dans le domaine militaire, ou encore pour effectuer des opérations de secours.

Le routage est une fonction primordiale dans les réseaux ad hoc où chaque nœud joue le rôle d'un routeur et participe activement dans la transmission des paquets de données. Ainsi, un nœud peut communiquer directement avec un autre s'il est dans sa portée radio. Sinon, il compte sur la coopération des voisins pour relayer ses messages. Cette fonction de routage est divisée en deux parties : i) *le protocole de routage* et ii) *le service de routage*. Un protocole de routage est un ensemble de règles qui spécifient la manière avec laquelle les routeurs communiquent entre eux pour échanger des informations sur la topologie leur permettant de construire leur propre vision du réseau. Un service de routage assure la transmission des paquets de données en utilisant la vision construite précédemment. Dans les réseaux ad hoc, l'utilisation des protocoles de routage classiques, utilisés pour les réseaux filaires est inappropriée puisqu'il faut prendre en considération les caractéristiques de ce type de réseaux (support de transmission, mobilité, ressources limitées, absence de centralisation, etc.). Ainsi, il a été nécessaire de concevoir des protocoles de routage spécifiques aux réseaux ad hoc.

Dans la suite de ce chapitre, nous présentons les concepts fondamentaux des algorithmes de routage ad hoc en nous basant sur quelques exemples (section 1.1). Ensuite, nous nous penchons dans la section 1.2 sur les attaques auxquelles les nœuds font face dans les réseaux ad hoc. Dans la section 1.3, nous présentons les solutions et les mécanismes de sécurité qui ont été proposés dans la littérature. Enfin, nous positionnons notre travail dans la section 1.4 par rapport à cet ensemble

de solutions.

1.1 Concepts fondamentaux des protocoles de routage ad hoc

Le routage est une opération essentielle dans les réseaux ad hoc puisqu'il constitue la base pour l'échange de données entre les nœuds mobiles. Selon Murthy et al. [MRM04], un protocole de routage ad hoc doit assurer :

- L'échange d'informations de routage garantissant l'établissement de chemin entre une source et une destination en se basant sur un certain nombre de critères (plus court, plus rapide, moins congestionné, absence de boucle de routage, etc.) ;
- La maintenance des chemins.

Cette description met en évidence deux processus essentiels à chaque protocole de routage ad hoc : i) *la découverte de route* et ii) *la maintenance des routes*.

Parallèlement, Murthy et al. [MRM04] insistent sur le fait de prendre en considération les facteurs suivants lors de la conception d'un protocole de routage ad hoc :

- La mobilité compte parmi les caractéristiques des nœuds dans un réseau ad hoc. Mais, cette liberté de mouvement n'est pas sans inconvénients : elle cause la rupture plus au moins fréquente des liens entre les nœuds ;
- Les nœuds ont des ressources limitées en terme d'autonomie de la batterie, de taille de la mémoire et de puissance de calcul ;
- Le canal de transmission sans fil a un taux d'erreur élevé (entre 10^{-5} et 10^{-3}) comparé au médium filaire (entre 10^{-12} et 10^{-9}) impliquant la perte de paquets. De plus, le canal est partagé par l'ensemble des nœuds présents dans cet espace ce qui implique que la bande passante des liaisons est limitée.

En fonction du mode de fonctionnement de la phase de découverte du chemin et de la mise à jour d'informations de routage, les protocoles de routage ad hoc sont classés en 3 catégories : pro-actif, réactif et hybride. Dans les trois sous-sections qui suivent, nous présentons chacune de ces catégories et fournissons des exemples représentatifs de protocoles.

1.1.1 Protocoles de routage ad hoc pro-actifs

Les protocoles de routage pro-actifs se basent sur l'établissement de routes à l'avance. Les nœuds mettent à jour périodiquement les données de routage de façon à obtenir en permanence le plus court chemin (calculé en terme du nombre de nœuds intermédiaires, aussi appelé nombre de sauts) vers tous les nœuds du réseau. Ainsi, si un nœud désire transmettre un paquet vers une destination, il consulte sa table de routage qui lui indique immédiatement le chemin à suivre. Il existe deux approches pour ce type de protocoles :

- L'approche vecteur de distance où chaque nœud diffuse les distances qui le séparent de tous les autres nœuds du réseau ;
- L'approche à état des liens où il s'agit de diffuser des descriptions des liens avec les nœuds voisins.

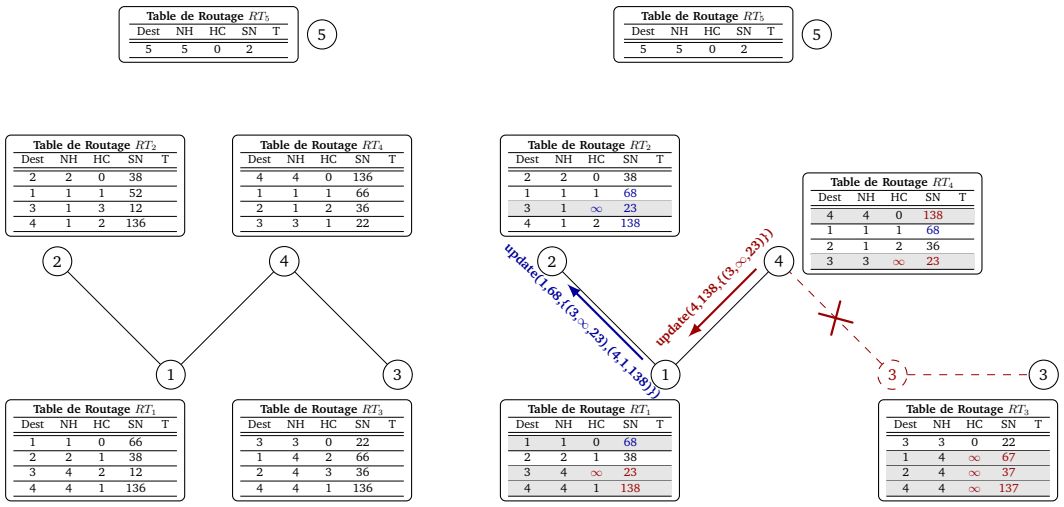
Dans ce qui suit, nous détaillons ces approches par l'intermédiaires de deux exemples de protocoles pro-actifs de routage ad hoc : DSDV [PB94] et OLSR [CJ03].

1.1.1.1 Destination Sequence Distance Vector (DSDV)

DSDV [PB94] (*Destination-Sequenced Distance-Vector*) est l'un des premiers protocoles de routage ad hoc pro-actifs à vecteur de distance. Il se base sur l'algorithme distribué Bellman-Ford DBF (*Distributed Bellman-Ford* [BGN87]) qui a été modifié pour s'adapter aux réseaux ad hoc.

Comme il s'agit d'un protocole pro-actif, chaque nœud a, à chaque instant une vision complète du réseau. Pour ce faire, chaque nœud récupère les distances le séparant de chaque autre nœud du réseau et ne garde que le plus court chemin. Ceci est fait grâce à des échanges périodiques d'informations sur leurs tables de routage respectives. Ces échanges sont classés en deux types :

– Les mises à jour incrémentales (*incremental updates*) pour lesquelles seules les données qui ont subit des modifications depuis la dernière mise à jour sont envoyées. Un exemple est présenté dans la figure 1.1 où, suite au déplacement du nœud 3 qui n'est plus à portée radio, le nœud 4 initie une procédure de mise à jour (*update*) qui ne concerne que l'entrée correspondant au nœud 3 dans sa table de routage (voir figure 1.1b). Chaque nœud recevant ce message le transfère en incluant les entrées qui viennent d'être modifiées. C'est le cas du nœud 1 qui initialise une mise à jour suite à la réception de celle du nœud 4 ;



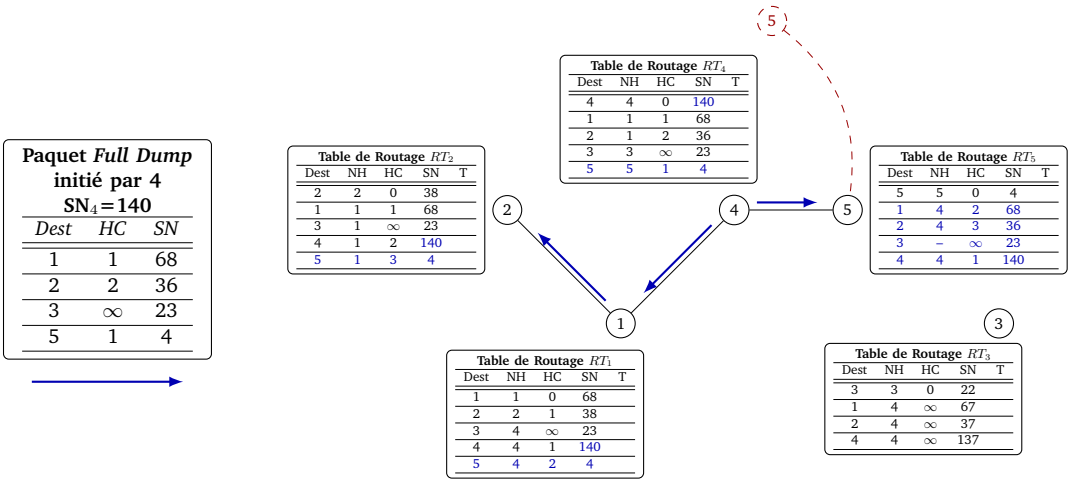
a. Après échanges de messages de mises à jour, les tables de routage se sont stabilisées

b. Modifications suite au déplacement du nœud 3

FIGURE 1.1 – Mise à jour incrémentale

– Les mises à jour complètes (*full dump*) pour lesquelles la totalité de la table de routage est envoyée. La figure 1.2 montre un exemple de cette procédure où le nœud 4 envoie la totalité de sa table de routage à tous les nœuds du réseau ce qui induit des changements au niveau de leurs tables de routage.

Outre son adresse et son propre numéro de séquence, chaque paquet de mise à jour doit contenir une liste des routes ajoutées/modifiées pour laquelle chaque entrée est un triplet formé par :



a. Paquet de mise à jour émis par le nœud 4

b. Propagation du paquet et répercussion sur les tables de routage

FIGURE 1.2 – Mise à jour complète (*full dump*)

l'adresse de la destination $Dest$, le nombre de sauts HC pour l'atteindre (*Hop Count*) et le dernier numéro de séquence connu associé à cette destination (*Sequence Number*) qui permet notamment de distinguer les nouvelles routes des anciennes et évite ainsi la formation de boucles de routage. La figure 1.2a montre un exemple de ce paquet de mise à jour.

Pour gérer la mobilité des nœuds, DSDV associe à chaque nœud un minuteur (`timer`) qui est mis à jour à la valeur maximale à chaque fois qu'un message est reçu du voisin : c'est un indicateur de validité du lien. Ainsi, lorsque ce minuteur expire, le nœud considère que le voisin en question n'est plus à porté radio et que le lien est rompu. Il peut aussi utiliser les messages de la couche 2 pour détecter les ruptures de liens. La détection d'un lien rompu se traduit au niveau de l'entrée correspondante dans la table de routage par l'assignement de la valeur ∞ au nombre de sauts (en pratique, il s'agit de n'importe quelle valeur supérieure au maximum autorisé) et l'incrémentation du numéro de séquence au prochain numéro impair [1]. Toutes les routes utilisant ce nœud qui n'est plus joignable sont aussi mises à jour comme étant des routes invalides. Ces changements sont envoyés en priorité à tous les voisins en utilisant un paquet de mise à jour. Il est à noter que c'est le seul cas où un nœud autre que la destination pourra changer le numéro de séquence de la destination qui n'est plus joignable (voir figure 1.1b cas des nœuds 3 et 4).

À la réception d'un paquet de mise à jour, les routes avec les plus grands numéros de séquences sont privilégiées pour le choix des routes puisque cela signifie une route plus fraîche. Dans le cas de numéros de séquences égaux, le plus court chemin est retenu en se basant sur le nombre de saut. Le nœud intermédiaire procède ensuite à la rediffusion des informations qu'il vient de modifier dans sa table de routage tout en incrémentant son numéro de séquence.

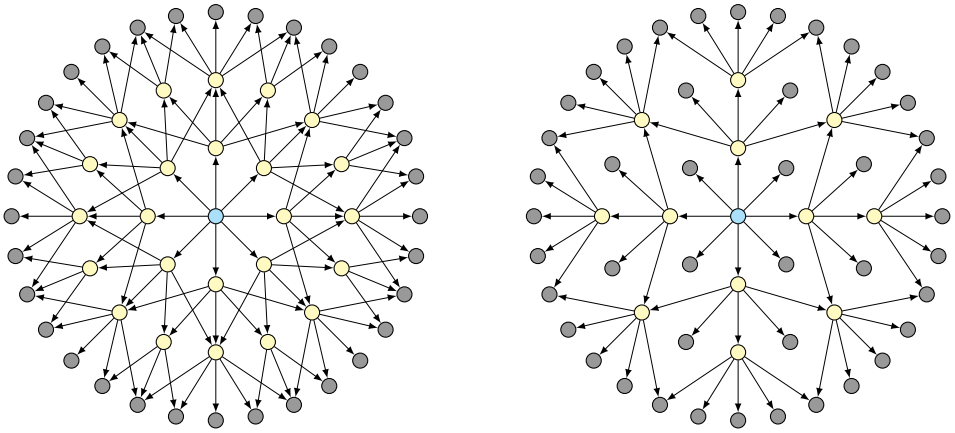
Malgré les améliorations qu'il propose par rapport à DBF en éliminant le problème des boucles de routage (*routing loops*) et le problème du comptage à l'infini (*counting to infinity*) grâce notamment à l'utilisation des numéros de séquence, DSDV reste long et coûteux. Il nécessite des mises à

1. Ce mécanisme est utilisé par DSDV pour différencier les routes valides ayant des numéros de séquence pairs, des routes invalides ayant des numéros impairs. Par exemple, un nœud détectant qu'un lien n'est plus disponible change le numéro qui est pair dans sa table de routage vers le numéro impair qui est immédiatement au dessus.

jour régulières de ses tables de routage même lorsque le réseau est inactif. À chaque mise à jour, un nouveau numéro de séquence est nécessaire ce qui augmente le temps avant que le réseau converge. Ceci rend DSDV peu adapté aux réseaux très dynamiques.

1.1.1.2 Optimized Link State Routing (OLSR)

OLSR [CJ03] (*Optimized Link State Routing*) est un protocole pro-actif à état de lien (*link state*). OLSR apporte certaines améliorations sur le principe de base de l'état de lien en vue d'atteindre de meilleures performances dans un contexte ad hoc : il permet de minimiser l'inondation du réseau en diminuant les retransmissions redondantes dans la même région du réseau et réduit la taille des paquets échangés. Pour cela, OLSR se base essentiellement sur la notion de relais multi-point (*MPR, Multi Point Relay*), un sous ensemble des voisins à un saut qui permet d'atteindre la totalité des voisins à deux sauts. Ainsi, lors d'une diffusion, tous les voisins reçoivent et traitent le message mais seulement les nœuds choisis comme MPR le retransmettent ce qui diminue considérablement le nombre de messages émis dans le réseau (voir figure 1.3).



a. Routage par inondation (24 transmissions pour atteindre tous les nœuds à 3 sauts)

b. Routage avec les nœuds MPR (12 transmissions pour atteindre tous les nœuds à 3 sauts)

FIGURE 1.3 – Avantage de l'utilisation des MPR

Principe de fonctionnement du protocole. OLSR étant pro-actif, chaque nœud construit en permanence une vision de la topologie du réseau sous forme d'un graphe où les arcs constituent les liens entre les nœuds. La cohérence de cette vision est assurée grâce à des diffusions périodiques des liens sortants. Ainsi, un nœud recevant ces informations met à jour sa vision de la topologie et applique l'algorithme du plus court chemin pour choisir le prochain saut en direction de chaque destination. Nous présentons maintenant les étapes permettant la construction de la topologie :

– **Écoute des voisins** (*neighbor sensing*) : Il s'agit du processus de découverte du voisinage direct et symétrique qui est effectué grâce à la diffusion périodique de messages de type HELLO contenant des informations sur le voisinage ainsi que l'état des liens (*link state*) le reliant à cet ensemble de nœuds. Ce message de contrôle est destiné exclusivement aux voisins à un saut et n'est donc pas retransmis. De cette manière, un nœud construit la liste des voisins à un saut (*Neighbor Set*) tout en marquant les liens symétriques et puisqu'il voit

aussi la liste des voisins de ceux-ci, il construit la liste des voisins à deux sauts 2HNS (*2-Hops Neighbor Set*).

– **Sélection des relais multi-point** : Cette sélection est faite de façon indépendante par chaque nœud. Elle passe par le choix du sous ensemble des nœuds à un saut qui permettent d'atteindre l'intégralité de voisins à deux sauts. Dans la figure 1.4, le nœud 1 choisit 2 comme MPR parce que c'est le seul nœud qui lui permet d'atteindre 5. Ensuite, il choisit le nœud 3 lui permettant d'atteindre 6, 7 et 8. De cette manière il couvre la totalité des voisins à deux sauts. Le sous ensemble obtenu est annoncé à tous les voisins dans des messages HELLO ultérieurs.

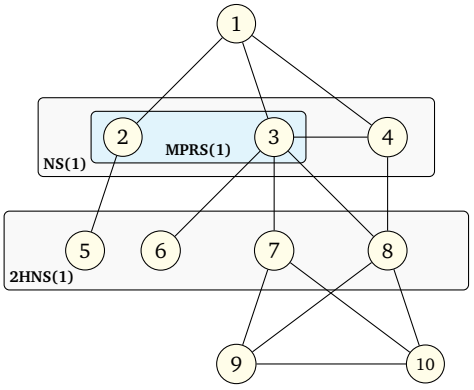


FIGURE 1.4 – Choix des relais multi-point pour le nœud 1

– **Déclaration des relais multi-point** : Les nœuds MPR diffusent des paquets de contrôle spécifiques appelés TC (*Topology Control*) pour construire une base d'informations sur la topologie du réseau. Les messages TC sont transmis à intervalles réguliers et déclarent l'ensemble MPRSS (*Multi Point Relay Selector Set*), c'est-à-dire l'ensemble contenant les voisins ayant choisit le nœud origine de ce message comme MPR. Les informations sur la topologie du réseau reçues dans les messages TC sont enregistrées dans la table de topologie (*topology table*).

– **Calcul de la table de routage** : La table des voisins (*neighbor table*) ainsi que la table de topologie (*topology table*) sont utilisées pour le calcul de la table de routage qui se base sur l'algorithme du plus court chemin [Dij59]. Toute modification de l'une de ces tables entraine la modification de la table de routage.

Cette amélioration à base de relais multi-point fournit des routes optimales en nombre de sauts tout en diminuant le nombre de messages de contrôles qui circulent lors d'une diffusion. Il convient ainsi aux grands réseaux ad hoc mais semble être moins efficace pour des petits réseaux [PJ03].

1.1.2 Protocoles de routage ad hoc réactifs

À l'inverse des protocoles pro-actifs, les protocoles réactifs ne construisent pas de tables de routage au préalable. Si un nœud a des données à transmettre, il entreprend une recherche de route et lorsque le chemin est établi, l'acheminement des données peut commencer. Cette recherche de route diffère d'un protocole à un autre selon l'approche utilisée : état de lien ou vecteur de distance. Nous présentons dans ce qui suit les protocoles DSR [JHM07, JM96] et AODV [PR99, PBRD03]

qui utilisent respectivement l'approche état de lien et l'approche vecteur de distance.

1.1.2.1 Dynamic Source Routing (DSR)

Le protocole DSR [JM96, JHM07] est un protocole de routage ad hoc réactif à état de lien. Les routes sont construites à la demande en utilisant le routage source : chaque nœud inclut son adresse dans l'entête de telle sorte qu'en arrivant à la destination, le paquet contient une liste complète et ordonnée de nœuds par lesquels le paquet a transité de la source à la destination (figure 1.5a). Cette liste est renvoyée à la source dans un paquet de réponse de route (figure 1.5b). Nous donnons plus détails sur le processus de découverte de route dans ce qui suit.

Découverte de route. Lorsqu'un nœud source désire envoyer des données à une destination et qu'il ne trouve pas de route disponible pour cette destination dans son cache (*route cache*), il initialise une demande de route RREQ (*Route REQuest*). C'est le cas du nœud 1 dans la figure 1.5a. La RREQ contient un identifiant unique (*route request identifier*), la destination à atteindre et une liste d'adresses de nœuds qui contient initialement uniquement l'adresse de la source (cette liste constituera le chemin entre la source et la destination à la fin du processus de découverte).

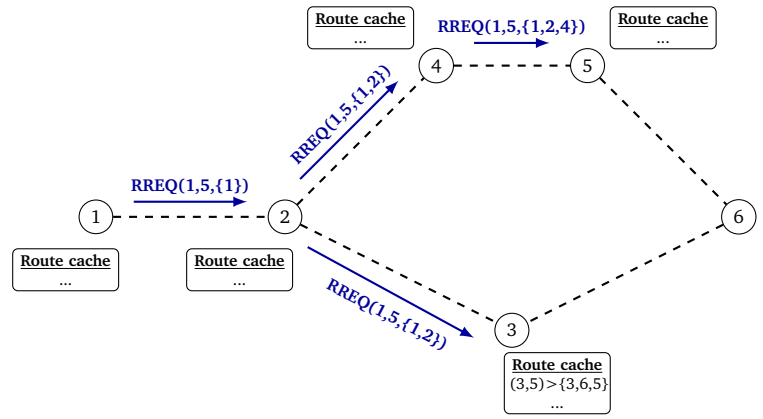
Lorsqu'un nœud intermédiaire reçoit la demande de route RREQ, il commence par vérifier s'il ne s'agit pas d'une requête déjà traitée en cherchant dans l'historique l'existence du couple identifiant cette RREQ. Si c'est le cas, le paquet est ignoré sinon, le nœud rajoute son adresse dans la liste du paquet et rediffuse ce paquet à son tour après l'avoir ajouté dans son historique. C'est le traitement que les nœuds 2 et 4 ont suivi dans l'exemple de la figure 1.5a.

Lorsque le paquet RREQ arrive à la destination, la liste contenue dans le paquet constitue le chemin complet pour l'atteindre (cas du nœud 5 dans la figure 1.5a). La destination crée alors une réponse de route RREP (*Route REPly*) en y copiant la liste contenue dans la RREQ reçue et en insérant son adresse à la fin de cette liste. Une fois envoyée, cette réponse de route suivra le chemin contenu dans la liste jusqu'à atteindre la source. Ainsi, le chemin est établi entre la source et la destination et la transmission de données peut débuter.

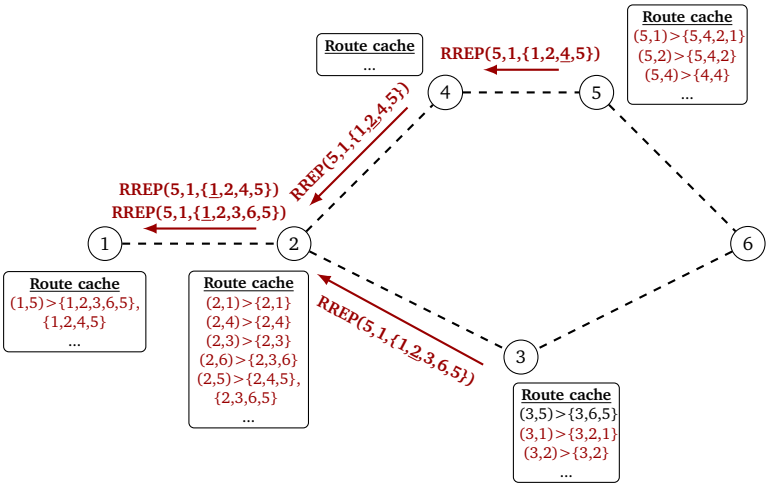
Dans certains cas, un nœud intermédiaire peut avoir une route qui mène à la destination dans son *route cache* (nœud 3 dans la figure 1.5). Dans cette situation, le nœud intermédiaire peut générer une réponse de route en concaténant le chemin qu'il a reçu dans le paquet RREQ avec celui qui se trouve dans son *route cache* en s'assurant qu'il n'y a pas de nœud qui figure dans les deux parties auquel cas il devra renoncer à la création de la RREP pour éviter la création des boucles de routage.

À la fin du processus de découverte de route, un nœud peut avoir dans son cache plus d'une route pour certaines destinations auquel cas il devra choisir une route en se basant sur le plus court chemin ou en utilisant une autre métrique (e.g. rapidité d'établissement du chemin).

Maintenance des routes. Lors de la transmission d'un paquet, chaque nœud est responsable de l'acheminement des données sur le lien en direction du prochain saut. Il devra s'assurer que les données sont bien parvenues au prochain saut. Un accusé de réception peut garantir la confirmation de la validité du lien (par exemple un accusé de niveau 2 ou en entendant la retransmission du destinataire ou le cas échéant un accusé spécifique à DSR). Si un nœud ne reçoit pas un accusé



a. Propagation de la demande de route



b. Propagation de la réponse de route

FIGURE 1.5 – Exemple du processus d'établissement de route entre 1 et 5

de réception suite à un envoi de paquet, il considère que le lien est rompu et supprime cette route du cache. Il crée alors un paquet erreur de route RERR (*Route ERRor*) qu'il envoie à tous les nœuds ayant envoyé un paquet sur ce lien depuis le dernier accusé de réception.

Un avantage de DSR est que les nœuds intermédiaires n'ont pas besoin de garder des informations sur les différents chemins pour pouvoir envoyer les paquets de données puisque ces routes sont incluses dans l'entête ce qui garantie l'absence de boucles de routage. À l'inverse, l'utilisation de DSR peut être coûteuse dans un réseau à forte mobilité et densité du trafic car il surcharge le réseau avec un grand nombre de messages de contrôle.

1.1.2.2 Ad hoc On-demand Distance Vector (AODV)

AODV (Ad hoc On demand Distance Vector) [PR99, PBRD03] est un protocole de routage réactif à vecteur de distance qui s'inspire de DSDV. Contrairement à celui-ci, il ne construit pas *a priori* la table de routage mais réagit à la demande et essaie de trouver un chemin avant de router les informations. Tant que la route reste active entre la source et la destination, le protocole

de routage n'intervient pas, ce qui diminue le nombre de paquets de routage échangés entre les nœuds constituant le réseau. Lorsqu'un nœud S essaie de communiquer avec un nœud D, l'échange de messages se fait en plusieurs étapes décrites ci-dessous à l'aide de l'exemple de la figure 1.6.

Découverte de route. Lorsqu'un nœud source a besoin d'une route vers une certaine destination (e.g. le nœud 1 dans la figure 1.6 désire envoyer des données au nœud 5) et qu'aucune route n'est disponible (la route peut être non existante, avoir expiré ou être défaillante), la source 1 diffuse en *broadcast* (voir figure 1.6a) un message de demande de route RREQ (*Route REQuest*). Ce message contient un identifiant (RREQ_ID) associé à l'adresse de la source (@SRC) qui servira à identifier de façon unique une demande de route. Le nœud 1 enregistre cet identifiant de paquet RREQ ([RREQ_ID, @SRC]) dans son historique (*buffer*) et l'associe à un *timer* qui décomptera sa durée de vie au delà de laquelle cette entrée sera effacée.

Quand un nœud intermédiaire (cas des nœuds 2 et 4 dans la figure 1.6b) qui n'a pas de chemin valide vers la destination reçoit le message RREQ, il ajoute ou met à jour le voisin duquel le paquet a été reçu. Il vérifie ensuite qu'il ne l'a pas déjà traité en consultant son historique des messages traités. Si le nœud s'aperçoit que la RREQ est déjà traitée, il l'abandonne et ne la rediffuse pas. Sinon, il met à jour sa table de routage à l'aide des informations contenues dans la requête afin de pouvoir reconstruire ultérieurement le chemin inverse vers la source. Il incrémente ensuite le nombre de sauts HC (*Hop Count*) dans la demande de route et la rediffuse.

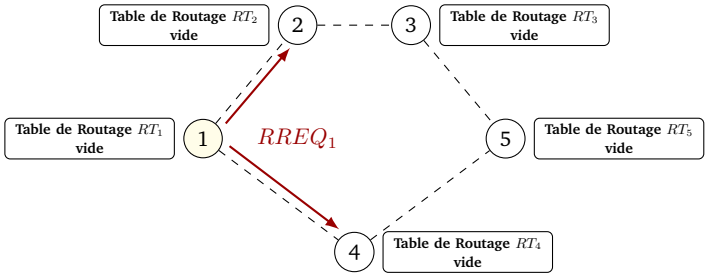
Il est à noter qu'AODV utilise le principe des numéros de séquence pour pouvoir maintenir la cohérence des informations de routage. Ce numéro, noté SN (*Sequence Number*), est un champ qui a été introduit pour indiquer la fraicheur de l'information de routage et garantir l'absence de boucles de routages.

À la réception d'un paquet RREQ (figure 1.6c), la destination 5 ajoute ou met à jour dans sa table de routage un chemin vers le nœud voisin duquel il a reçu le paquet (nœud 4) ainsi qu'un chemin vers la source 1. La destination 5 génère ensuite une réponse de route RREP qu'elle envoie en *unicast* vers le prochain saut en direction de la source (voir figure 1.6c). Notons qu'un nœud intermédiaire peut aussi générer un RREP si la requête l'autorise à le faire (bit *destination_only* de la RREQ mis à 0) et qu'il dispose déjà dans sa table de routage d'un chemin valide vers la destination 5.

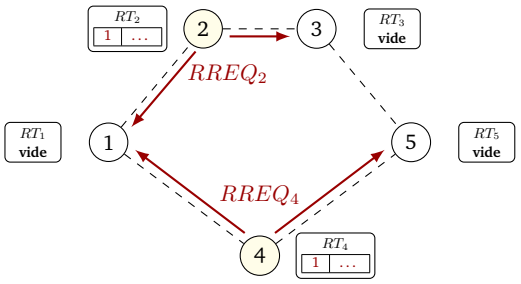
Les nœuds intermédiaires qui reçoivent la RREP (cas du nœud 4 dans la figure 1.6d) vont mettre à jour le chemin qui mène à la destination dans leurs tables de routage et retransmettre en *unicast* le message (après avoir incrémenté le nombre de sauts) vers le nœud suivant en direction de la source sachant que cette information a été obtenue lors du passage de la RREQ.

Lorsque la réponse de route atteint la source (nœud 1 dans l'exemple), un chemin bidirectionnel est établi entre la source et la destination (voir figure 1.6e) et la transmission de paquets de données peut débuter.

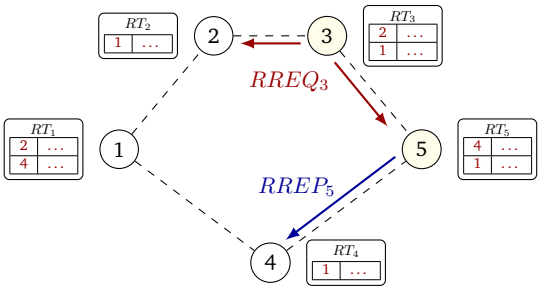
Maintenance des routes. Afin de maintenir les routes, une transmission de messages HELLO est effectuée. Ces messages sont en fait des réponses de route (RREP) diffusés aux voisins avec un nombre de sauts égal à un. Si au bout d'un certain temps, aucun message n'est reçu d'un nœud voisin, le lien en question est considéré défaillant. Alors, un message d'erreur RERR (*Route ERRor*) se propage vers la source et tous les nœuds intermédiaires vont marquer la route comme invalide



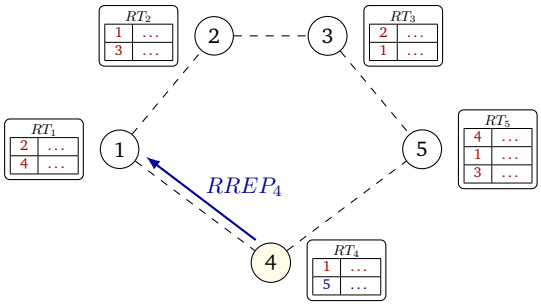
a. Le nœud 1 initialise une demande de route pour obtenir un chemin vers 5



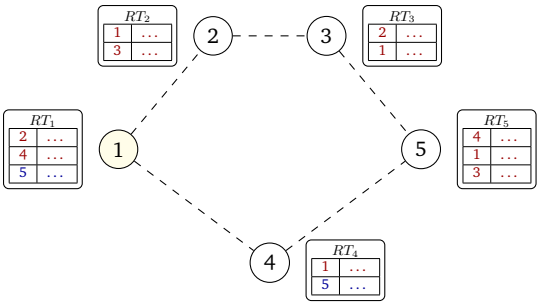
b. 2 et 4 retransmettent la demande de route



c. 5 initialise une réponse de route



d. 4 retransmet la réponse de route



e. Arrivée de la réponse de route à la source 1

FIGURE 1.6 – Exemple d'établissement de route entre 1 et 5

et au bout d'un certain temps, l'entrée correspondante est effacée de leur table de routage. Le message d'erreur RERR peut être diffusé ou envoyé en *unicast* en fonction du nombre de nœuds à avertir de la rupture de liaison détectée. Ainsi, s'il y en a un seul, le message est envoyé en *unicast* sinon, il est diffusé.

AODV a l'avantage de réduire le nombre de paquets de routage échangés étant donnée que les routes sont créées à la demande et utilise le principe du numéro de séquence pour éviter les boucles de routage et garder la route la plus fraîche. Cependant, l'exécution du processus de création de route occasionne des délais importants avant la transmission de données.

1.1.3 Protocoles de routage ad hoc hybrides

La combinaison d'un protocole réactif et d'un protocole pro-actif donne lieu à une troisième catégorie qu'on appelle les protocoles hybrides. Ce type de protocoles adopte une méthode pro-active pour établir les chemins à l'avance dans un voisinage ne dépassant pas quelques sauts (2

ou 3 sauts) et utilise une méthode réactive au delà de cette limite. La combinaison de ces deux techniques partage le réseau en zones où un nœud peut décider directement à la réception d'un message si la destination fait partie de la même zone ou non, auquel cas il devra rediriger le message vers une autre zone. Dans cette section, nous présentons l'un de ces protocoles, ZRP (*Zone Routing Protocol* [Haa97, HPS02d]).

Zone Routing Protocol est historiquement le premier protocole de routage ad hoc hybride. Proposé en 1997, il inclut tel que définie dans [HPS02d] :

– Un protocole de routage pro-actif (IARP [HPS02c] *Intrazone Routing Protocol*) qui fournit une vue détaillée du voisinage à k-sauts (par exemple k=2) appelé zone de routage (*routing zone*). Pour pouvoir construire cette zone, chaque nœud a besoin du voisinage à un saut qui est obtenue grâce au protocole de la couche liaison ou en utilisant un protocole prévu à cet effet comme le NHDP [CDD10] (*Neighborhood Discovery Protocol*). La figure 1.7 présente les zones de routage pour les nœuds 1 et 4. Ces deux zones se chevauchent du fait que chaque nœud maintient sa propre zone de routage.

  Les protocoles de routage pro-actifs à état de lien peuvent être modifiés et utilisés comme un IARP en limitant les mises à jour de l'état de lien au rayon de la zone de routage. De cette manière, un nœud peut décider immédiatement lors de la réception d'un paquet s'il a un chemin vers la destination ou non et peut ainsi répondre au nom de tous les nœuds de la zone à laquelle il appartient. Ce qui évite l'effort d'explicitement interroger le reste des nœuds de la même zone.

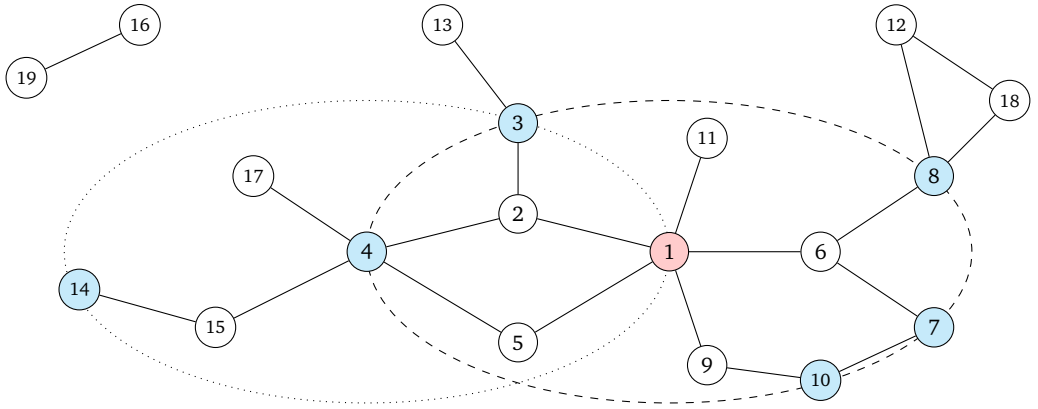


FIGURE 1.7 – Zone de routage de rayon=2 du nœud 1 et 4

– Si un nœud n'a pas de chemin vers la destination, l'IERP [HPS02b] (*IntErzone Routing Protocol*) prend le relais pour la recherche de routes en dehors de la zone dans laquelle le nœud se trouve. Ainsi, une demande de route est créée et envoyée aux nœuds périphériques de la zone de routage (opération appelée *bordercast*). Par exemple, les nœuds 3, 4, 7, 8 et 10 sont des nœuds périphériques de la zone de routage du nœud 1 dans la figure 1.7.

  Les demandes de route créées sont acheminées vers la périphérie de la zone en utilisant le protocole BRP [HPS02a] (*Bordercast Resolution Protocol*). Ce dernier se base sur la topologie obtenue grâce à l'IARP pour la construction d'un arbre multicast (*bordercast tree*) donnant les différents chemins pour atteindre les nœuds périphériques d'une zone. Ces nœuds périphériques vérifient à leur tour l'existence de destination dans leurs zones et si c'est le cas,

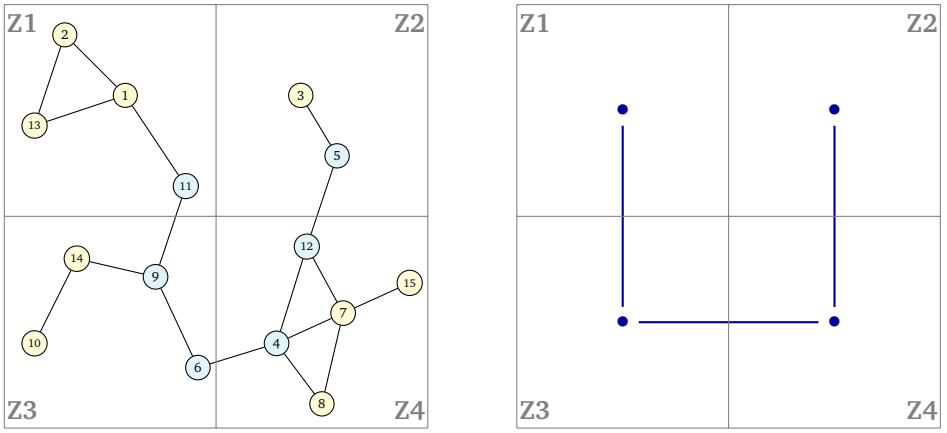
un paquet de réponse de route est retourné à la source. Dans le cas contraire, ils diffusent la demande de route à leurs propres nœuds périphériques qui, à leur tour, effectuent le même traitement. Chaque nœud retransmettant la demande de route fait attention à ne pas retransmettre la demande de route aux nœuds l'ayant déjà traité afin d'optimiser le temps de découverte de route et éviter les boucles de routage.

Pour résumer, lorsqu'un nœud implémentant ZRP veut joindre une destination, il commence sa recherche dans la zone à laquelle il appartient. S'il la trouve, il peut l'atteindre immédiatement étant donnée que cette entrée est maintenue dans son cache grâce au protocole pro-actif (IARP). Sinon, il envoie une requête de demande de route aux nœuds périphériques grâce au protocole réactif (IERP) en utilisant BRP pour la livraison de ces demandes. Ainsi, les nœuds périphériques recevant la demande de route recherchent dans leurs zones respectives et ainsi de suite jusqu'à atteindre la destination. Ce protocole présente l'avantage de diminuer le nombre de messages de contrôle qui transitent sur le réseau comparé aux protocoles pro-actifs ou réactifs. De plus, il permet de diminuer le temps de latence pour trouver de nouvelles routes.

D'autres protocoles hybrides existent mais rares sont ceux qui ont abouti à au moins un draft déposé dans l'IETF [2] (*Internet Engineering Task Force*) à l'instar de ZRP. Nous citons à titre d'exemple ZHLS [JN99] qui se base sur la division du réseau en zones géographiques fixes (voir figure 1.8) et chaque nœud peut se localiser dans une zone grâce à ses coordonnées GPS (*Global Positioning System*). Ainsi, la topologie du réseau est divisé en deux niveaux :

– Niveau intra-zone présentant les liens qui existent entre les nœuds appartenant à la même zone et les liens éventuels vers des nœuds d'autres zones (figure 1.8a). Ceci est matérialisé par une table de routage intra-zone au niveau de chaque nœud.

– Niveau inter-zone présentant le schéma de connexion entre les zones (figure 1.8b) et qui est matérialisé par une table de routage inter-zone au niveau de chaque nœud établissant ainsi la carte des zones.



a. Topologie niveau intra-zone b. Topologie niveau inter-zone

FIGURE 1.8 – Topologie dans un réseau implémentant ZHLS

2. L'IETF est un groupe international qui participe dans l'élaboration de standards pour l'Internet. Cet organisme produit la plupart des nouveaux standards d'Internet.

Pour atteindre une destination, un nœud commence par consulter sa table de routage intra-zone. S'il ne trouve pas la destination, il initialise une demande de localisation qui permet de retrouver la zone à laquelle appartient le nœud recherché. Ainsi, le chemin menant à la destination est connu puisqu'on a la zone à laquelle elle appartient et le transfert des données peut débuter.

En conclusion, ZHLS est un protocole de routage ad hoc hybride qui se base sur la décomposition du réseau en zones géographiques. Il maintient à jour pour chaque nœud les tables de routage intra-zone et inter-zone et utilise une technique réactive pour retrouver la zone à laquelle appartient un nœud qu'il veut joindre.

Après avoir présenté les différents types de protocoles de routage ad hoc par l'intermédiaire d'exemples représentatifs, nous nous intéressons, dans la section suivante, à la sécurité de ces protocoles en présentant notamment les attaques auxquels ils font face.

1.2 Sécurité du routage dans les réseaux ad hoc

Le routage constitue une fonction principale dans les réseaux ad hoc. Il s'agit du mécanisme par lequel les chemins sont créés pour acheminer les données à la bonne destination à travers un réseau. Cette fonction est divisée en deux parties : le *protocole de routage* et le *service de routage*.

- Le protocole de routage est un ensemble de règles qui spécifient la manière avec laquelle les routeurs communiquent entre eux pour échanger des informations sur la topologie leur permettant de construire leur propre vision du réseau.
- Le service de routage assure quant à lui la transmission des paquets de données en utilisant la vision construite précédemment.

Nous constatons que ces deux définitions sont complémentaires : sans le protocole de routage, il n'y aurait plus de vision du réseau ce qui empêche l'acheminement des données vers la destination, partie assurée par le service de routage. Nous nous intéressons dans la suite à la sécurisation des protocoles de routage et non pas de celle du service de routage. Ce choix est justifié par le fait que sécuriser le protocole de routage revient à sécuriser le résultat de cette étape qui est la vision du réseau ce qui permet de sécuriser le service de routage. Il est important de noter que nous ne discutons pas la sécurité des paquets de données échangés mais celle des paquets du protocole de routage.

Pour assurer la sécurisation d'un système d'information, il faut assurer trois propriétés [ITS91] : l'intégrité, la confidentialité et la disponibilité. Puisque nous nous intéressons à la sécurité des protocoles de routage, nous proposons de définir ces trois propriétés dans ce contexte.

- **Confidentialité.** Cette propriété assure que des entités non autorisées n'accèdent pas aux informations échangées et traitées dans les messages de routage et ne portent pas atteinte à la confidentialité de la topologie. Assurer cette propriété est en relation directe avec le domaine d'application du réseau ad hoc. C'est le cas éventuellement du domaine militaire où la topologie ne doit pas être divulguée à des fins stratégiques.
- **Disponibilité.** Cette propriété implique que les services ou les ressources demandées soient disponibles en temps opportun, même s'il y a un problème ou un dysfonctionnement dans le système. Dans notre contexte, il s'agit de garantir un accès au service de routage à n'importe quel moment. Cependant, ceci nécessite la construction préalable d'une vision du réseau.

– **Intégrité.** Cette propriété garantit que les messages de routage échangées entre les entités n'ont pas été altérées par des entités non autorisées. Ceci préserve par conséquent l'intégrité de la table de routage construite à partir de ces messages.

De notre point de vue, assurer l'intégrité des messages de routage échangées entre les nœuds constitue la tâche la plus importante pour assurer la sécurité d'un protocole de routage ad hoc. Ceci permet de préserver l'intégrité de la table de routage qui sert de base pour assurer le service de routage.

La RFC 4949 [Shi07] (*Internet Security Glossary*) définit une attaque comme étant un acte intentionnel par lequel une entité tente de se soustraire à des services de sécurité et de porter atteinte à la politique de sécurité d'un système. C'est en fait toute circonstance ou évènement susceptible de nuire à un système grâce à un accès non autorisé, la destruction, la divulgation ou la modification de données, ou le déni de service.

Les protocoles de routage ad hoc tel que conçus manquent de contrôles de sécurité, ce qui augmente le risque d'attaques qui peuvent être orchestrées par des nœuds externes ou internes en tenant compte du positionnement du nœud malhonnête par rapport au réseau.

– Les attaquants **externes** sont des nœuds qui ne font pas partie du réseau. Dans ce cas, la mise en place de mécanismes cryptographiques peut résoudre le problème : seuls les nœuds ayant les autorisations nécessaires pourront accéder au réseau ou déchiffrer le contenu.

– Les attaquants **internes** sont des nœuds faisant légitimement partie du réseau. Ces nœuds ont les autorisations et le matériel cryptographique nécessaires pour appartenir au réseau et les autres nœuds leur font *a priori* confiance.

Les attaquants internes sont plus difficiles à détecter et à éviter que les attaquants externes. Elles requièrent de mettre en place des mécanismes de détection des nœuds attaquants et des mécanismes pour contrer leurs agissements. Nous appelons ces attaquants « nœuds malhonnêtes » dans la suite de ce document.

Quelque soit sa position (interne ou externe), le nœud malhonnête utilise plusieurs techniques pour perturber le bon fonctionnement du protocole. La combinaison de ces techniques peut aboutir à une attaque plus élaborée. Ces techniques que nous qualifions d'***élémentaires*** sont présentées ci-après :

– Rejeu de messages : le nœud malhonnête enregistre une séquence de trafic qu'il réinjecte ensuite dans le réseau.

– Modification de messages : le nœud malhonnête modifie un ou plusieurs champs du message avant de le retransmettre.

– Suppression (ou effacement) de messages : le nœud malhonnête supprime des messages.

– Fabrication de messages : le nœud malhonnête fabrique un message et l'injecte dans le réseau.

Il est à noter que ces techniques d'attaques peuvent paraître dans certaines références comme étant des attaques *actives* tentant d'altérer les ressources du système ou d'affecter leurs activités.

Nous nous intéressons dans cette thèse aux attaques élémentaires puisqu'elles constituent la base d'un comportement malhonnête. Nous présentons dans ce qui suit à des fins d'illustration une liste non exhaustive des attaques sur les protocoles de routage ad hoc présentées dans la littérature. Lors de ces attaques, les nœuds malhonnêtes se basent sur une ou plusieurs techniques

(rejeu, modification, suppression, fabrication) pour s'attaquer aux messages de contrôle échangés entre les nœuds afin d'atteindre leur objectif : interrompre, perturber, absorber, ou attirer le trafic et même provoquer un déni de service. Nous regroupons ces actions malhonnêtes selon la propriété de sécurité à laquelle ils s'attaquent. Cependant, il est à noter que certaines attaques peuvent violer plus d'une propriété.

Attaques contre la confidentialité. L'action de l'attaquant peut se résumer à écouter ou surveiller les transmissions. Le but est d'obtenir les informations qui transitent soit directement soit après une analyse. Ce type de comportement est très difficile à détecter puisque le nœud malhonnête n'altère pas les messages échangés et ne participe pas en envoyant des messages supplémentaires.

Dans d'autres cas, l'attaquant peut provoquer la révélation de certaines informations. C'est le cas de l'attaque par divulgation d'emplacement (*location disclosure attack* [VGS+04]) où le nœud malhonnête tente d'obtenir l'identité des nœuds formant le chemin à une destination. Il est à noter que cette attaque interne concerne plutôt des protocoles réactifs où un nœud ne connait que le prochain saut en direction de la destination comme AODV (il serait inutile d'appliquer cette attaque sur DSR par exemple puisque le chemin entre la source et la destination est inclus dans le paquet).

Attaques contre la disponibilité. Les nœuds malhonnêtes peuvent s'attaquer à la disponibilité des nœuds. Une technique consiste à consommer leurs ressources en sollicitant de manière continue le nœud cible (en fabriquant des messages inutiles à destination de la cible ou en modifiant des paquets pour que les messages passent par la cible) ce qui provoque des traitements supplémentaires. C'est le cas de l'attaque par consommation des ressources (*Resource consumption attack* [SAT05]) ou l'attaque par privation de sommeil (*sleep deprivation torture* [SA99]). Une autre technique consiste à faire déborder la table de routage (*routing table overflow* [GS03]) de la cible. Ceci passe par la création de demandes de routes ou de messages de mise à jour pour des nœuds non existants jusqu'à débordement de la table de routage de la cible ce qui empêchera la formation de routes réelles.

Les nœuds égoïstes sont un type particulier de nœuds malhonnêtes qui ne participent pas volontairement au routage afin de conserver leur énergie. Ces nœuds peuvent porter atteinte à la disponibilité du routage puisque cela peut empêcher la formation de routes. Certes, ils n'attaquent pas activement le réseau, mais leur effet se fait ressentir en terme d'efficacité et de fluidité de communication. Une étude plus détaillée sur ce type de comportement est présenté dans [OKRK03].

Attaques contre l'intégrité. En s'attaquant à l'intégrité des messages, un nœud malhonnête fausse la table de routage. Son action repose sur des actions élémentaires.

Par exemple, l'attaque du trou de ver (*wormhole attack* [HPJ06]) se base sur le rejeu de messages (voir figure 1.9). Elle nécessite la coopération de deux ou plusieurs nœuds et la mise en place d'un tunnel de communication privé entre eux où les paquets entendus d'un côté sont retransmis de l'autre côté. Le tunnel créé peut être une liaison physique directe ou une liaison virtuelle comme c'est le cas dans [SDL+02] où les attaquants ont recourt à l'encapsulation des messages.

Les nœuds malhonnêtes peuvent modifier le contenu d'un message ou fabriquer du trafic non légitime. Les paquets injectés ou modifiés proposent par exemple des routes plus courtes,

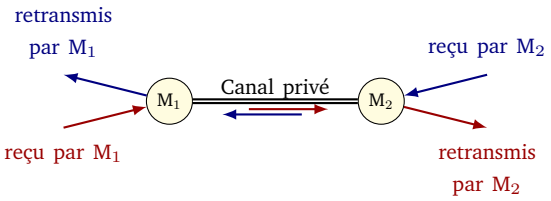
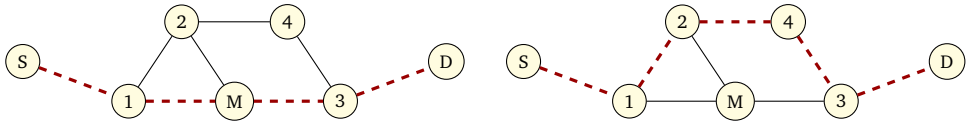


FIGURE 1.9 – Attaque trou de ver

plus fraiches, de nouvelles routes non-existantes ou même des routes à supprimer. C'est le cas dans [DLA02, KW03] où tout le trafic est redirigé vers un nœud (l'attaquant) qui absorbe totalement ou sélectivement le trafic ou aussi l'attaque par pollution de la table de routage (*routing table poisoning* [CKS+06]) qui à travers l'injection de faux messages provoque un routage non optimal, une congestion du réseau, ou une division du réseau. L'attaque par détour (*detour attack* [GAB06]) est un autre exemple où le nœud malhonnête fait en sorte que les routes ne passent plus par lui (voir figure 1.10).



a. Chemin obtenu entre S et D avant l'attaque

b. Chemin obtenu entre S et D après l'attaque

FIGURE 1.10 – Attaque par détour

L'attaquant peut aussi exploiter une propriété intéressante de certains protocoles : il s'agit du fait que le chemin par lequel la demande de route est parvenue en premier est retenue pour le routage. Ainsi, en exécutant la *Rushing attack* [HPJ03], l'attaquant retransmet plus rapidement les messages pour que la route qui passe par lui soit retenue. Une fois le nœud malhonnête sur la route, il peut absorber totalement ou sélectivement le trafic. Ces comportements ont été présentés dans [DLA02, KW03].

La combinaison des techniques précédemment présentées (rejeu, modification, suppression, fabrication) avec l'usurpation d'identité (*impersonation*) donne une autre panoplie d'attaques : l'attaquant usurpe l'identité et les privilèges d'un autre nœud en changeant son adresse IP ou MAC avec celle d'un nœud légitime. L'usurpateur pourra ainsi effectuer par exemple l'attaque *man-in-the-middle* [TBK+03] où il usurpe l'identité de la destination vis à vis de la source et de la source vis à vis de la destination sans qu'aucun d'eux (source ou destination) réalise qu'il est attaqué. Un autre exemple est l'attaque de *Sybil* [Dou02] où le nœud malhonnête revêt plusieurs identités et se comporte comme s'il était un ensemble de nœuds.

Le tableau 1.1 présente un récapitulatif des attaques expliquées tout au long de cette section. Il est à noter que la liste des attaques décrites bien que représentative n'est pas exhaustive. Nous constatons tout au long de cette analyse d'attaques que les nœuds malhonnêtes abusent de la confiance qui leur est accordé par l'entourage pour violer les propriétés de sécurité. Dans ce qui suit, nous nous intéressons donc principalement aux propositions qui ont été faites pour pallier aux problèmes de nœuds malhonnêtes dans les réseaux ad hoc.

Nom de l'attaque		Influence sur la propriété de sécurité			Technique			Motivation/Résultat
		Confidentialité	Disponibilité	Intégrité	Rejeu	Modification	Fabrication	
Écoute et analyse du trafic		√			Pas d'actions			– Intercepter des informations.
Location disclosure	[VGS+04]	√				√		– Découvrir l'emplacement des nœuds.
Resource consumption	[SAT05]		√				√	– Épuisement de la batterie.
Sleep deprivation torture	[SA99]		√				√	
Routing table overflow	[GS03]		√				√	– Débordement de la table de routage.
Selfish behavior	[OKRK03]		√		Pas d'actions			– Conserver son énergie.
Wormhole attack	[HPJ06]			√	√			– Création d'un tunnel ;
	[SDL+02]			√		√		– Perturber/Attirer le routage.
Detour attack	[GAB06]			√		√		– Ne pas participer dans le routage ; – Augmenter délai de bout en bout ; – Dégrader les communications.
Black hole attack	[DLA02]			√		√	√	– Absorber le trafic.
	[KW03]			√		√	√	
Routing table poisoning	[CKS+06]			√			√	– Routage non optimal ; – Congestion du réseau ; – Division du réseau.
Rushing attack	[HPJ03]			√	Exécution plus rapide			– Attirer le trafic.
Man-in-the-middle attack	[TBK+03]	√	√	√		√	√	– Utiliser l'usurpation d'identité.
Sybil attack	[Dou02]	√	√	√	√	√	√	– Identités multiples (créées et usurpées).

TABLE 1.1 – Récapitulatif des attaques présentées

1.3 Solutions et mécanismes de sécurité

De nombreuses solutions ont été proposées pour sécuriser les protocoles de routage ad hoc. Nous classons ces solutions en deux catégories : (i) les *systèmes de sécurité pro-actif*, dans le sens où des mécanismes sont établis à l'avance pour assurer la sécurité en renforçant la résistance du système aux attaques grâce notamment aux solutions à base de cryptographie et (ii) les *systèmes de sécurité réactifs* qui réagissent (adaptation/prise de décision immédiate) selon le comportement du voisinage et qui est lui même divisé en solutions de gestion de *réputation* et de *confiance* (voir figure 1.11).

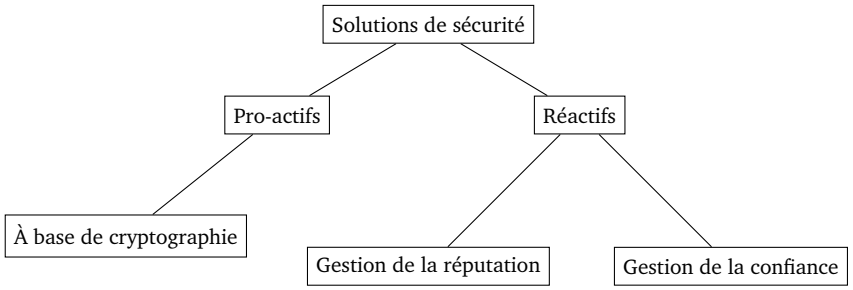


FIGURE 1.11 – Principales pistes de recherche

Dans chacune des sous sections suivantes, nous détaillons les solutions présentées dans la littérature pour chaque axe de recherche.

1.3.1 Solutions basées sur la cryptographie

Les solutions basées sur la cryptographie sont souvent utilisées contre les attaquants externes. Elles font la distinction entre les nœuds qui sont autorisés à prendre part au réseau (qui sont supposés se comporter correctement) et les nœuds qui n'y sont pas autorisés et qui sont considérés *a priori* comme étant des attaquants. La plupart de ces solutions se basent sur des mécanismes de chiffrement et de signature numérique pour assurer l'authentification des nœuds et la confidentialité et l'intégrité des messages : l'annexe A propose un bref aperçu de ces principes cryptographiques. Nous présentons dans ce qui suit les propositions utilisant les mécanismes cryptographiques pour sécuriser les protocoles de routage ad hoc.

Certaines approches s'appuient sur des chaines de hachés cryptographiques pour assurer l'authentification. C'est le cas de SEAD [HJP03], un protocole de routage pro-actif basé sur DSDV qui protège contre les attaques par modification du numéro de séquence et du nombre de sauts dans les messages de mise à jour. Les auteurs proposent que chaque nœud calcule une suite de hachés $(h_0, h_1, h_2, \ldots, h_n)$ en appliquant une fonction de hachage H tel que :

$$\begin{cases} h_0 = x & où \quad x \; valeur \; initiale \; choisie \; aléatoirement \\ h_i = H(h_{i-1}) & avec \quad 0 < i \leqslant n \end{cases}$$

Cette suite est organisée en segments de m éléments chacun :

$$(h_0,\, h_1,\, h_2,\, \ldots,\, h_{m-1}),\, (h_{km},\, h_{km+1},\, h_{km+2},\, \ldots,\, h_{km+m-1}),\, \ldots,\, h_n$$

où $k = \frac{n}{m} - i$, m étant le diamètre maximal du réseau et i le numéro de séquence. Il est à noter

que la valeur h_n est distribuée aux autres nœuds de telle sorte que chacun aura le haché final de tous les autres nœuds qu'il stocke dans l'entrée correspondante de la table de routage. La table 1.2 présente un exemple de la création de la chaine de haché.

i \ j	0	1	2	3	4
1	h_{15}	h_{16}	h_{17}	h_{18}	h_{19}
2	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}
3	h_5	h_6	h_7	h_8	h_9
4	h_0	h_1	h_2	h_3	h_4

TABLE 1.2 – SEAD : Calcul des chaines de haché pour assurer l'authentification dans un réseau de diamètre $m = 5$, i est le numéro de séquence, j est le nombre de saut et la longueur de la chaine de haché est $n = 20$

Pour un numéro de séquence i et un nombre de sauts j, la valeur h_{n-mi+j} correspond au haché à ajouter dans le message de mise à jour. Ceci permet au récepteur de vérifier l'authenticité de l'émetteur du message de mise à jour en appliquant successivement la fonction H, $mi - j$ fois sur le haché reçu et de comparer la valeur obtenu à celle stockée dans la table de routage (h_n). Dans l'exemple de la table 1.2 et pour les valeurs suivantes : numéro de séquence $i = 1$ et nombre de saut $j = 2$, l'émetteur du message de mise à jour joint le haché h_{17}. Le récepteur quant à lui calcule $H(H(H(h_{17})))$ et compare le résultat à la valeur stockée dans la table de routage pour authentifier l'émetteur. De cette manière, un attaquant ne peut jamais diminuer le nombre de sauts ou augmenter le numéro de séquence ce qui permet de protéger contre ce type d'attaques. Cependant, SEAD ne considère pas la modification d'autres champs comme le prochain saut ou la destination. Il ne protège pas non plus contre la fabrication et l'envoi d'un nouveau message de mise à jour à destination d'un autre nœud en utilisant la même métrique et le même numéro de séquence qu'un message de mise à jour récent. Ainsi, un nœud malhonnête peut modifier les champs qui ne sont pas protégés ou fabriquer et injecter des messages où les données qui seront vérifiées sont récentes alors que le reste est faux.

Les mêmes auteurs proposent aussi une extension à DSR appelée ARIADNE [HPJ05]. Cette solution assure une authentification point-à-point des messages de routage en utilisant les fonctions de hachage à clé secrète (HMAC *Hash-based Message Authentication Code*). Cependant, pour assurer une authentification sécurisée, ARIADNE se base sur TESLA [PCTS02], un protocole pour assurer l'authentification sécurisée lors des diffusions. Chaque nœud a une clé secrète qui lui permet de calculer une chaine de hachés qui est utilisée par la suite de la manière suivante : lorsqu'il transmet un message de demande de route (*Route Request*), le nœud lui ajoute un HMAC calculé sur l'ensemble du message avec le dernier haché encore non-utilisé de la chaine. Si un message de réponse de route (*Route Reply*) passe par le nœud, celui-ci révèle la pré-image de la valeur qu'il avait utilisée dans le message *Route Request*. Lorsque tous les nœuds sur le chemin réalisent cette opération, le chemin est authentifié. Pour fonctionner, ARIADNE requiert que tous les nœuds du réseau soient synchronisés (suite à l'utilisation de TESLA) et que chacun d'entre eux connaissent la dernière valeur de la chaine de hachés de tous les autres. Cette extension sécurise le protocole contre les attaques par modification et par fabrication mais reste encore vulnérable aux comportements égoïstes.

Adjih et al. [ACJ+03] proposent M-SOLSR (*Message based Secure OLSR*) qui utilise la signature pour assurer que le message vient d'un nœud de confiance et assurer ainsi l'authentification de la

source. Les messages de contrôle sont aussi horodatés pour éviter le rejeu d'anciens messages. Le réseau est divisé en deux : les nœuds de confiance et les nœuds inconnus non-sûrs. Cette approche se limite à la détection des nœuds inconnus qui se comportent mal et supposent un comportement honnête des nœuds de confiance. Pour pallier à cette limitation, les auteurs proposent une technique avancée de signature dans [RACM04] : un nouveau message ADVSIG (*ADVanced SIGnature*) est ajouté à la spécification d'OLSR permettant de prouver l'existence d'un lien avec le nœud émetteur du message et ainsi détecter les nœuds malhonnêtes considérés de confiance.

Avec ARAN [SDL+02], les auteurs ne se contentent pas seulement de l'authentification des nœuds de bout en bout, ils proposent un service de non-répudiation en utilisant des certificats pré-établis distribués par un serveur de confiance. Chaque nœud transmettant un message de demande de route doit le signer, ce qui augmente la taille des messages de routage à chaque saut. La figure 1.12 montre un exemple d'échange de messages lors du processus de découverte de route : le nœud source S désire connaître le chemin vers D. Il initie alors le paquet de demande de route $\{Request, D, Cert_S, N_S, t\}_{k_{priv(S)}}$. Ce paquet de demande de route ($Request$) à destination de D contient le certificat de l'émetteur $Cert_S$, une valeur aléatoire N_S et un horodatage t. Ce paquet est signé à l'aide de la clé privée de la source $k_{priv(S)}$. Le premier voisin recevant le paquet (dans l'exemple, il s'agit du nœud A) vérifie la validité de la signature et la validité du certificat de S avant de signer le message avec sa signature et son certificat ($k_{priv(A)}, Cert_A$). Chaque nœud intermédiaire (2-sauts ou plus) vérifie la signature et le certificat du nœud duquel il a reçu le message et les remplacent par sa signature et son certificat et ainsi de suite jusqu'à ce que le message atteigne la destination. L'inconvénient de cette méthode est qu'elle utilise l'authentification saut par saut en vérifiant à chaque fois le certificat ce qui augmente considérablement le calcul au niveau de chaque nœud ainsi que la taille des messages, ce qui consomme plus de bande passante.

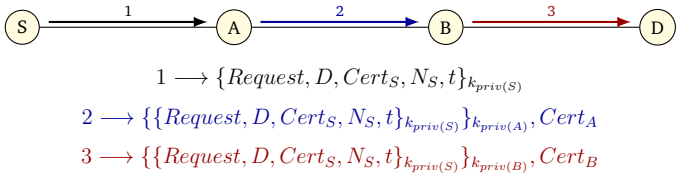


FIGURE 1.12 – Authentification lors de la découverte de route à chaque saut dans ARAN

L'extension SAODV [ZA02] proposée par Zapata et al. combine l'utilisation de chaines de hachés et de signatures pour garantir l'authentification de la source et l'intégrité des messages. Contrairement à ARAN dans lequel les messages sont signés à chaque retransmission, seule la source signe les messages et ce uniquement sur les champs qui ne changent pas afin de limiter le coût en CPU et la taille des messages. La chaine de hachés protège les champs variables à la manière de SEAD.

Les solutions présentées dans cette section se basent sur des mécanismes cryptographiques. Des primitives cryptographiques sont utilisées pour assurer l'authentification (du message, du chemin ou de la source), l'intégrité et la non-répudiation. Ces solutions permettent à des nœuds qui ont les autorisations nécessaires de participer au réseau et empêchent les autres d'interférer. Cependant, l'utilisation de ces mécanismes suppose l'existence d'une autorité centralisée (pour la certification ou la distribution de clés), ce qui n'est pas toujours évident dans un réseau ad hoc. De plus, ces mécanismes n'empêchent pas certains nœuds malhonnêtes de servir leurs intérêts par exemple en fournissant délibérément de fausses informations aux autres nœuds. Les solutions proposant de

palier à cette limitation se classent en deux catégories : les systèmes de gestion de réputation et les systèmes de gestion de confiance.

1.3.2 Systèmes de gestion de la réputation

Certaines approches se sont intéressées aux problèmes de coopération entre les nœuds qui est la base du fonctionnement du routage dans les réseau ad hoc. Les solutions proposées se basent sur des systèmes de réputation. Mui et al. [MMH02] définissent la réputation comme étant la perception qu'un nœud a d'un autre à propos de ses intentions, au vu d'un ensemble d'anciennes actions (observations antérieures) et qui peut être combinée à d'autres points de vue (recommandations). Ainsi, par son comportement un nœud ne pourra pas décider de sa réputation mais pourra influencer sa propre réputation. Une valeur numérique est associée à la réputation d'un nœud dont le calcul diffère d'une solution à une autre mais qui se base essentiellement sur la surveillance du voisinage et sur les informations de première main et/ou de seconde main obtenues.

L'un des premier travaux [MGLB00] propose d'utiliser un *watchdog* et un *path rater* pour atténuer les conséquences d'un comportement malhonnête. Alors que le premier permet de surveiller le voisinage et de détecter la non transmission d'un paquet par un voisin, le second permet la gestion de la réputation et du routage en évaluant chaque chemin utilisé ce qui permet d'éviter ceux comportant des nœuds malhonnêtes. Les nœuds ne comptent que sur leur *watchdog* et ne s'échangent aucune information de réputation entre eux pour éviter les attaques par fausses accusations ce qui augmentera le délai de détection. Cependant ceci permet aux nœuds malhonnêtes, même s'ils sont détectés par certains nœuds, de continuer à utiliser les ressources du réseau.

Ce mécanisme de surveillance du voisinage (*watchdog*) apparait par la suite dans la majorité des travaux de recherche. Ainsi, Michiardi et al. proposent CORE [MM02], un mécanisme de réputation collaboratif (*Collaborative Reputation mechanism*) qui utilise un *watchdog*. Dans cette solution, les auteurs divisent la réputation en trois catégories : la réputation subjective basée sur les observations, la réputation indirecte qui repose sur les évaluations positives des voisins et la réputation fonctionnelle qui correspond à un comportement spécifique à une tâche. Les valeurs obtenues de chaque catégorie sont pondérées pour calculer une valeur unique de la réputation qui sera utilisée pour la prise de décisions concernant la coopération ou l'isolation graduelle d'un nœud. Bien que les nœuds ne s'échangent que les réputations positives, cette proposition reste vulnérable à des attaques par de faux éloges rendant un nœud malhonnête plus difficile à démasquer.

À l'inverse de CORE, le mécanisme d'inférence sensible au contexte (*Context-aware inference mechanism* [PW02]) se base sur l'échange d'évaluations négatives liées à un processus de découverte de route unique sur une période de temps bien déterminée. Les accusations reçues des autres nœuds combinées à une connaissance approximative de la topologie sont évaluées par le système d'inférence pour décider de la culpabilité d'un nœud. L'accusation doit provenir de plusieurs nœuds faute de quoi le seul nœud qui revendique l'accusation est accusé lui même de malhonnêteté. Ceci décourage les fausses accusations mais potentiellement aussi les accusations correctes de peur d'être le seul dénonciateur, ce qui réduit la dissémination des informations.

CONFIDANT [BLB02, BLB05] (*Cooperation Of Nodes Faireness In Dynamic Ad hoc NeTworks*) propose de combiner l'utilisation des évaluations positives et négatives. Ainsi, chaque nœud surveille le comportement de ses voisins tout en gardant trace des estimations (positives ou négatives) de la réputation (opinion qu'il forme) ainsi que des opinions des voisins. Ces estimations sont par la suite combinées pour donner un niveau de confiance dans un chemin selon les nœuds par lesquels

le paquet sera acheminé. Tout évènement suspicieux provoque une mise à jour de la réputation qui ne doit pas dépasser un seuil au delà duquel les routes contenant ce nœud sont supprimées et un message d'alarme est envoyé aux voisins pour transmettre les avertissements. Chaque nœud recevant l'alarme va vérifier si l'alarme provient d'une source de bonne réputation ainsi que le nombre total d'alarmes reçues concernant ce nœud malhonnête avant de la prendre en considération.

TAODV [LLL04] (*Trusted AODV*) étend AODV en lui ajoutant un modèle de gestion de la réputation, un protocole de routage spécifique pour les informations de réputation et un système de gestion de clés. Chaque nœud maintient à propos de chaque autre nœud une opinion ω_B^A sous la forme d'un triplet (b_B^A, d_B^A, u_B^A) formé des valeurs de confiance (b), de méfiance (d) et d'incertitude (u) sachant que $b_B^A + d_B^A + u_B^A = 1$. Lors de l'initialisation du réseau, toutes les opinions sont à $(0, 0, 1)$: il s'agit d'une totale incertitude sur l'honnêteté et la crédibilité des autres nœuds. Après une phase d'échange de clés, l'interaction commence dans le réseau impliquant des évènements positifs et négatifs. Selon le résultat de l'échange, le compteur correspondant pour chaque nœud est incrémenté (p pour positif et n pour négatif). Ces compteurs sont stockés au niveau de la table de routage et serviront à la formation des opinions.

Pour résumer, les systèmes de gestion de réputation s'emploient à protéger les protocoles de routage ad hoc contre les attaquants internes ou externes en mettant en œuvre un mécanisme de surveillance du voisinage (*watchdog*) et en calculant une valeur numérique (opinion) représentant le comportement de chaque voisin. Cette valeur est ensuite comparée à un seuil pour estimer la fiabilité d'un nœud ou d'une route. Selon la technique utilisée, cette opinion est calculée à partir d'observations directes ou indirectes. Ces systèmes sont souvent utilisés pour détecter les nœuds égoïstes et les forcer à coopérer et à participer dans les opérations de routage. Leur inconvénient majeur reste leurs vulnérabilités aux attaques par fausses accusation.

1.3.3 Systèmes de gestion de la confiance

La notion de confiance est souvent confondue abusivement selon nous à la réputation même s'il existe des associations entre les deux. On peut définir la confiance comme étant la croyance et la conviction dans l'honnêteté, la sincérité, la franchise, la compétence, la fiabilité et la crédibilité d'une entité ou d'un service. Cette notion est omni-présente dans les protocoles de routage ad hoc où les nœuds se font mutuellement confiance et se basent sur l'hypothèse d'un comportement honnête. Ainsi, les solutions à base de systèmes de gestion de confiance tirent avantage des propriétés intrinsèques des protocoles afin de détecter les comportements malhonnêtes. Dans un sens, comme pour les systèmes de gestion de réputation, ces systèmes se comportent comme un système de détection d'intrusion essayant de créer une deuxième ligne de défense contre les attaquants internes.

Wang et al. [WLMG05] proposent une approche de détection de nœuds malhonnêtes en se basant sur la vérification des propriétés sémantiques inhérentes aux spécifications du protocole OLSR. Ces propriétés sémantiques sont utilisées par chaque nœud pour détecter les incohérences par rapport au comportement correct qui est définit par la spécification d'OLSR. La redondance d'informations lors de l'échange de messages HELLO et TC permet de vérifier l'existence d'informations contradictoires. Le système de détection présenté repose sur les propriétés suivantes :

– L'ensemble des nœuds déclarés dans le message TC_X d'un nœud X est toujours inclut ou égal dans l'ensemble des nœuds déclarés dans le message $HELLO_X$ du même nœud X.

- Si un nœud X reçoit un message TC_Y le déclarant comme sélecteur MPR alors il faut que le nœud ayant initié le message soit dans le voisinage de X ($Y \in HELLO_X$).
- Si un nœud X reçoit un message TC_Y le déclarant comme sélecteur MPR alors X doit avoir précédemment choisi le voisin (Y) duquel le message est parvenu, comme *MPR* dans son message $HELLO_X$
- Un nœud qui a envoyé un message TC est supposé entendre la retransmission à l'identique de son message par les nœuds qu'il a choisi comme *MPR* (le seul changement étant l'adresse de la destination dans l'entête IP).

Les propriétés présentées dans cette approche ne concernent que la vision locale et directe de chaque nœud et ne permet pas de partager les observations. De plus, les auteurs n'ont pas spécifié les mesures à prendre pour stopper et isoler les nœuds malveillants.

Cuppens et al. [CCBNR07] se basent sur la même technique utilisant les propriétés intrinsèques du protocole OLSR pour détecter les inconsistances dans les messages de contrôle échangés (*HELLO* et *TC*). L'approche repose sur le recoupement des informations redondantes lors de l'échange de ces messages pour détecter les profils des nœuds (coopératif, paresseux, égoïste, réservé, honnête, menteur, diffamateur). Ceci passe par la vérification du comportement du voisin. Cette redondance d'informations sur la topologie implique des propriétés qui sont utilisées pour vérifier si un nœud a menti et ainsi pouvoir agir en conséquence :

- Générer une alerte,
- Ne pas choisir le menteur comme *MPR*,
- Ne plus envoyer au menteur des messages,
- Gérer attentivement les informations suspectes.

Cette approche propose des mesures contre les nœuds malveillants mais elle reste elle aussi limitée à la vision locale et ne vérifie pas la cohérence de toutes les observations. De plus, il n'y a pas de vérification de cohérence avec les propriétés sémantiques des nouvelles routes calculées suite à une contremesure.

Une approche alternative est proposée par Adnane et al. [ASBM07]. Les auteurs se basent sur les spécifications formelles des relations de confiance implicites dans OLSR. Cette approche souligne le processus de construction de la confiance qui repose sur des clauses du type : « une entité A a confiance en une entité B à propos de certains aspects ». Ce langage permet d'exprimer les attaques liées à la violation de ces relations de confiance et offre en outre des indicateurs permettant aux nœuds de détecter ceux ayant un comportement malhonnête. L'approche d'Adnane et al. met en avant la méfiance envers les informations reçues du réseau en intégrant un raisonnement au niveau de chaque nœud pour la détection d'anomalies. Des mesures de prévention et de contremesures [ABS09] ont été mises en place pour résoudre certaines vulnérabilités du protocole et traiter les comportements anormaux. La prévention permet de valider le voisinage avant d'établir des relations de confiance. Ainsi, chaque nœud doit fournir la preuve de l'existence d'un lien particulier (preuve de voisinage) et vérifier les preuves de voisinage fournies par les voisins. Les contremesures sont quant-à elles prévues pour isoler et punir un nœud malveillant (suite à une attaque), mais aussi pour partager cette détection en envoyant les preuves mettant en cause le nœud en question.

Une autre approche de détection que nous pouvons citer est celle proposée par Huang et

Lee [HL04]. Elle se base sur la vérification de l'exécution normale du protocole et tout comportement qui ne suit pas la spécification est considéré anormal. Les auteurs utilisent un EFSA (*Extended Finite State Automaton*) pour modéliser le comportement normal dans AODV. Ils construisent un modèle de comportement composé par des états, des transitions et des actions permettant d'inspecter la logique d'exécution lorsque certaines conditions sont remplies cherchant toute violation des spécifications. L'EFSA est aussi associé à l'utilisation d'algorithmes d'apprentissage statistique pour détecter les actions anormales. L'analyse des attaques préalablement effectuée par les auteurs montre qu'ils se basent sur des actions élémentaires facilement traduites en violations de l'EFSA. Ces violations sont dues à : (i) *un état non valide* dans la spécification, (ii) *une transition incorrecte*, ou (iii) *une action inattendue* suite à l'exécution incorrecte ou inachevé d'une action.

Cette proposition peut être considérée comme un système de gestion de la confiance même si elle n'en est pas un : à travers la modélisation de l'EFSA, nous pouvons retrouver les règles mettant en œuvre une confiance entre les nœuds. Ce travail, associé à celui de Ning et Sun [NS03], met en avant une approche de décomposition d'attaques en actions élémentaires permettant de clarifier l'action de l'attaquant sur la spécification du protocole.

1.4 Positionnement

La section 1.3 présente un aperçu des différentes solutions qui ont été mises en place pour sécuriser les protocoles de routage ad hoc. Ces solutions sont classées en trois catégories :

- Les solutions utilisant les mécanismes cryptographiques assurent l'authenticité et l'intégrité des messages. Toutefois, ces mécanismes sont d'une part assez lourds à mettre en place notamment lors de la distribution des clés, et d'autre part, ils n'empêchent pas des nœuds ayant le matériel cryptographique nécessaire de se comporter malhonnêtement. Ces solutions sont donc plutôt destinées à fournir une protection contre les attaques externes.
- Les solutions à base de systèmes de réputation essaient de construire une opinion sur les autres nœuds sous forme de valeur numérique comparée à un seuil souvent obtenu par des méthodes heuristiques. Toutefois, un nœud peut avoir une bonne réputation puisqu'il participe aux opérations du réseau, tout en se comportant mal car il diffuse de fausses informations.
- Les solutions à base de systèmes de gestion de confiance essaient de profiter des propriétés intrinsèques des protocoles de routage pour chercher des incohérences entre les messages reçus. Ces solutions opèrent comme des systèmes de détection d'intrusions contre les attaquants internes.

Nous nous intéressons plus particulièrement à la dernière catégorie : les systèmes de gestion de confiance. La notion de confiance est plus sollicitée dans le contexte des réseaux ad hoc vu les caractéristiques particulières de ce type de réseaux (mobilité, distribué, auto-organisé, ad hoc) où les nœuds communiquent entre eux sans connaissance préalable de l'identité ou même de la topologie. Il nous semble judicieux de permettre à chaque nœud du réseau de mettre en place un mécanisme qui lui permet de raisonner sur les comportements des autres. Ce raisonnement utilise les observations qu'il collecte au fur et à mesure d'échange de messages et, en les corrélant, lui permet de détecter des incohérences caractéristiques d'actions malhonnêtes.

La majorité des solutions à base de systèmes de gestion de confiance ont été effectuées sur le protocole de routage ad hoc pro-actif OLSR. Nous sommes convaincus qu'il est intéressant d'appliquer une telle approche sur un protocole réactif. Notre choix s'est porté sur AODV, l'un des protocoles réactifs les plus répandus. Il a de plus fait l'objet d'une RFC numéro 3561 [PBRD03] (*Request For Comments*) qui stipule d'ailleurs qu'AODV a été conçu pour être utilisé dans des réseaux où les nœuds peuvent tous se faire confiance (« *AODV is designed for use in networks where the nodes can all trust each other.* » [PBRD03], page 6). Ainsi, les nœuds doivent pouvoir avoir confiance en les informations qu'ils reçoivent.

En ce sens, nous proposons de mettre en place un mécanisme permettant à chaque nœud participant à un réseau AODV de distinguer les nœuds dignes de confiance des nœuds malhonnêtes en se basant sur les observations qu'il a collecté. Et pour pouvoir distinguer les nœuds malveillants, nous adaptons la technique de décomposition d'attaques en actions élémentaires permettant de clarifier l'action de l'attaquant. Ce qui nous permet de trouver les règles facilitant la détection des incohérences. Dans la suite de cette thèse, nous présentons l'analyse de la confiance implicite effectuée sur le protocole de routage ad hoc AODV. Nous présentons ensuite le raisonnement qu'un nœud doit adopter pour détecter les incohérences dans les annonces.

CHAPITRE 2

ANALYSE DES ASPECTS DE CONFIANCE DANS AODV

La confiance est un concept fondamental sur lequel se base tout type de transaction, d'interaction et de communication. Elle peut être accordée à une personne physique ou morale (institution, système). Dans un réseau ad hoc ouvert, un nœud compte sur la coopération de nœuds qu'il ne connait pas pour pouvoir échanger des informations. Ceci suppose de faire l'hypothèse d'accorder sa confiance aux autres et aux informations qu'ils lui présentent.

La notion de confiance a été traité à maintes reprises dans la littérature. Cependant, il ne semble pas exister de définition faisant autorité, chacune abordant la confiance selon des points de vue différents :

- Du point de vue psychologique, la définition la plus répandue est celle de Deutsch [Deu62] qui considère que la confiance est basée sur une perception individuelle. Pour une même situation, chaque entité perçoit la situation différemment. L'auteur associe deux variables pour chaque cas à traiter : V_a+ pour un évènement bénéfique et V_a- dans le cas contraire.
- Du point de vue sociologique, Luhmann [Luh79] affirme que le concept de confiance est un moyen de réduction de la complexité de la société. Elle joue un rôle important dans les interactions qui y prennent part. Bok [Bok99] lui aussi considère la confiance comme un bien social qu'il faut protéger à tout prix.

  > *. . . trust is a social good to be protected just as much as the air we breathe or the water we drink. When it is damaged, the community as a whole suffers ; and when it is destroyed, societies falter and collapse.*
  >
  > *D'après Bok [Bok99], page 26-27.*

- Du point de vue mathématique, Gambetta [Gam00] considère la confiance comme un niveau

particulier de la probabilité subjective avec laquelle un agent évalue qu'un autre agent ou un groupe d'agents effectue une action particulière. Elle devient ainsi quantifiable (variant entre 0 et 1). Marsh [Mar99] trouve plus judicieux de s'intéresser au comportement de la confiance plutôt qu'à la confiance. Il propose un modèle théorique et assez complexe qui reste générique et non applicable à des cas réels.

Plusieurs travaux, à l'instar de [BAN90, GNY90, Ran92, YKB93, BBK94, Mar99, WLMG05, CCBNR07, ASBM07], présentent des modèles formels de calculs et de gestion de la confiance dans différents domaines. Dans notre travail, nous nous intéressons au comportement de la confiance dans les protocoles de routage ad hoc puisque chaque nœud est supposé se conduire correctement, impliquant ainsi une confiance implicite. Le comportement correct des nœuds est obtenu à partir de la spécification d'AODV (RFC 3561 [PBRD03]) et l'analyse de celle-ci nous permet de dégager les règles permettant la construction de la confiance entre les nœuds. Cependant, les nœuds malhonnêtes profitent des faiblesses du protocole pour servir leurs intérêts. Nous considérons que ces attaques exploitent la confiance aveugle que les nœuds développent envers leur voisinage, d'où la nécessité d'introduire de la méfiance. Ceci peut être accompli à travers l'analyse du trafic des voisins qui permet de détecter tout comportement anormal.

Dans ce qui suit, avant de présenter l'analyse que nous proposons pour la détection de comportement malhonnêtes, nous commençons par passer en revue les failles de sécurité relatives au protocole de routage ad hoc AODV qu'un nœud malveillant peut utiliser. Cette analyse nous permet de comprendre le comportement de l'attaquant.

2.1 Vulnérabilités du protocole de routage ad hoc AODV

Dans cette section, nous nous intéressons de plus près aux attaques sur le protocole de routage ad hoc AODV. Nous présentons comment les nœuds malhonnêtes opèrent pour aboutir à leur objectif. Ces comportements malveillants se basent essentiellement sur une ou plusieurs actions élémentaires. Les travaux de Ning et Sun [NS03] et Huang et Lee [HL04] renforcent cette constatation et proposent deux approches d'analyse d'attaques basées sur la décomposition d'attaques complexes en un ensemble de composants élémentaires appelées respectivement *anomalous basic events* et *atomic misuse*. Ce type d'approches est particulièrement intéressant dans le sens où il permet de clarifier l'action de l'attaquant sur la spécification du protocole.

Dans [NS03], les auteurs commencent par identifier les objectifs des attaquants. Ils en énumèrent quatre à savoir : (1) perturbation de route, (2) invasion de route, (3) isolation de nœud, ou (4) consommation des ressources. Ils étudient ensuite comment atteindre ces objectifs en agissant sur les messages de contrôle.

Ning et Sun classifient les actions malhonnêtes sur AODV en deux catégories : (i) *atomiques*, résultant de la manipulation d'un seul message de routage et (ii) *composées*, définies comme étant une collection d'actions atomiques. Les manipulations effectuées sur un message de routage peuvent être :

- Effacer un paquet ;
- Modifier un ou plusieurs champs du paquet avant de le retransmettre ;
- Fabriquer une réponse à la réception d'une demande de route RREQ ;
- Fabriquer activement des paquets de routage sans même avoir reçu de messages de routage.

Ces actions élémentaires sont appliquées aux messages de contrôle d'AODV pour tester l'effet produit et en déduire quel objectif est atteint. Le tableau 2.1 récapitule les actions malveillantes sur les messages de routage en fonction de l'objectif.

Action élémentaire		**Perturbation de route**	**Invasion de route**	**Isolation nœud**	**Consommation des ressources**
RREQ	Effacer	Oui [1]	Non	Non	Non
	Modifier	Oui	Oui	Partiel	Non
	Fabriquer A.	Oui	Oui	Partiel	Non
RREP	Effacer	Oui [1]	Non	Non	Non
	Modifier	Oui	Oui	Non	Non
	Fabriquer R.	Oui	Oui	Non	Non
	Fabriquer A.	Oui	Oui	Non	oui
RERR	Effacer	Oui [1]	Non	Non	Non
	Modifier	Oui	Non	Non	Oui
	Fabriquer A.	Oui	Non	Non	Oui

[1] dans certains cas

TABLE 2.1 – Récapitulatif des attaques élémentaires présentés dans [NS03]

Dans [HL04], les auteurs s'intéressent au processus de routage et ils le divisent en une série d'événements élémentaires. Chaque ensemble peut contenir une ou plusieurs opérations exécutées dans un ordre particulier. Par exemple, (1) la réception d'un paquet, (2) la modification des champs du paquet, et (3) la retransmission du paquet constituent un ensemble de trois opérations. La succession normale des évènements (ensembles) est présentée dans la spécification du système. Les auteurs utilisent une machine à états finis étendue pour représenter cette spécification où chaque transition correspond à un évènement élémentaire. Ainsi, tout comportement qui ne suit pas la spécification du système est considéré anormal.

Les auteurs proposent de caractériser tout événement élémentaire anormal par la cible à laquelle il s'attaque (message de routage, paquet de données ou table de routage) et par l'opération effectuée. Les différentes combinaisons entre cible et opération sont représentées dans le tableau 2.2. Ce tableau montre aussi à quelle propriété de sécurité chaque attaque porte atteinte.

Propriété de sécurité		**Événement par cible**		
		Messages de routage	**Paquets de données**	**Table de routage**
Confidentialité		Divulgation d'emplacement	Divulgation de données	-
Intégrité	Ajout	Fabrication	Fabrication	Ajouter route
	Suppression	Interruption	interruption	Supprimer route
	Changement	Modification *Rushing*	Modification	Changer le coût des liaisons
Disponibilité		Inondation	Inondation	Débordement de la RT

TABLE 2.2 – Taxonomie des événements élémentaires anormaux [HL04]

Nous présentons dans ce qui suit une liste (non-exhaustive) des attaques à base d'actions élémentaires sur le protocole de routage ad hoc AODV. La vision que nous adoptons est en partie la combinaison de ce que les auteurs dans [NS03, HL04] ont présenté : nous testons l'effet des actions élémentaires sur chaque message de routage sans se préoccuper de l'objectif à atteindre comme c'est le cas dans [NS03] et pour chaque attaque nous présentons la cible à la manière de [HL04].

2.1.1 Attaques élémentaires portant sur les demandes de route

Nous traitons dans ce qui suit l'effet de chaque action élémentaire sur les paquets de demande de route.

2.1.1.1 Suppression d'une demande de route

Un nœud malhonnête pourrait simplement **effacer** la demande de route reçue. En appliquant ce genre de comportement à tout message RREQ reçu, l'attaquant ne participe pas au routage : c'est comme s'il ne fait pas partie du réseau. Une autre variante serait d'effacer sélectivement des messages RREQ. Ce comportement peut être comparé à celui d'un nœud égoïste.

2.1.1.2 Modification d'une demande de route

À la réception d'une demande de route, le nœud malhonnête **modifie** un ou plusieurs champs qu'il n'est pas supposé modifier avant de retransmettre le message. La modification peut aussi porter sur un champs qu'il a le droit de modifier, mais il ne respecte pas la spécification pour le faire.

Plusieurs champs impliquent des traitement différents lorsqu'ils sont modifiés. Par exemple, le champs identifiant de la RREQ associé à l'adresse de la source permet d'identifier de manière unique une demande de route et indique la fraicheur de la demande de route. Puisqu'un nœud n'accepte que le première copie de RREQ, en augmentant cet identifiant, le nœud malhonnête peut garantir l'acceptation et le traitement de la RREQ modifiée par les autres nœuds.

Le nœud malhonnête peut jouer aussi sur le numéro de séquence de la source et/ou le nombre de saut dans une RREQ en augmentant le premier et en diminuant le second : pour assurer l'absence de boucles de routage, un nœud ne met à jour un chemin vers la source que si le numéro de séquence reçu dans la demande de route est plus grand que celui stocké dans la table de routage ou les numéros de séquence sont égaux et le nombre de sauts reçu plus petit que celui stocké dans la table de routage. Ainsi, l'intervention d'un nœud malhonnête sur ces champs provoque des changement dans la table de routage des voisins : le prochain saut en direction de la source devient le nœud malhonnête duquel le message RREQ modifié est reçu.

Le tableau 2.3 montre les champs de la demande de route où l'attaquant peut intervenir.

Champs de la RREQ	Est modifiable ?	Modification	Effet
Type	Non	Changer le type	Rejet du message (non conforme au type déclaré)
Identifiant	Non	Augmenter	Rendre la RREQ acceptable (plus fraiche)
		Diminuer	Rendre la RREQ non-acceptable
Nombre de sauts	Oui (autorisé +1)	Diminuer	Mettre à jour le chemin inverse vers la source
		Augmentation (>1)	Ralentir la découverte de route
Adresse destination	Non	Changer	Fausser la découverte de route (trafic inutile)
Adresse source	Non	Changer	
Numéro de séquence source	Non	Augmenter	Mettre à jour le chemin inverse vers la source
		Diminuer	Ne pas mettre à jour le chemin vers la source
Drapeau G	Non	inverser (0|1)	Envoyer ou supprimer la RREP gratuite
Drapeau D	Non	inverser (0|1)	Seulement la destination a le droit de répondre sinon tout nœud intermédiaire

TABLE 2.3 – Modifications possibles sur les champs des RREQ

2.1.1.3 Fabrication d'une demande de route

Les attaques décrites précédemment (section 2.1.1.1 et 2.1.1.2) sont déclenchées par la réception d'une demande de route. En revanche, les attaques par **fabrication** peuvent être effectués sans avoir reçu de messages RREQ. Le nœud malhonnête a besoin de collecter certaines informations, en écoutant le trafic par exemple, avant d'injecter le message fabriqué. Il est à noter que l'attaquant peut falsifier autant de champs qu'il veut générant l'effet discuté un peu plus haut dans la section 2.1.1.2. Il est à noter qu'une exécution répétitive de ce type d'attaque peut provoquer l'**inondation** du réseau par des messages de routage inutiles. Le nœud malhonnête peut orienter le trafic vers une seule destination ou faire croire que le trafic part d'une seule source ou les choisir (source/destination) au hasard.

2.1.1.4 *Rushing* d'une demande de route

Dans d'autres cas, le nœud malhonnête peut utiliser la technique du ***rushing*** [HPJ03] qui consiste à diminuer le temps de traitement des messages RREQ et les retransmettre plus rapidement de telle sorte qu'ils atteignent plus rapidement la destination. Ceci garantira pour le nœud malhonnête une place sur le chemin.

2.1.2 Attaques élémentaires portant sur les réponses de route

La mise en place de certaines attaques portant sur les réponses de route dépend de l'emplacement de l'attaquant : il faut qu'il soit choisit sur la route ou qu'il soit voisin d'un nœud faisant partie du chemin pour pouvoir entendre la transmission de la RREP, qui rappelons le est effectuée en *unicast*. Nous présentons dans ce qui suit l'effet de chaque action élémentaire sur les réponses de route.

2.1.2.1 Suppression d'une réponse de route.

Ce type d'attaque n'a un sens que si le nœud malhonnête a été choisie sur la route reliant la source à la destination. Dans ce cas, la **suppression** de la réponse de route empêche la formation du chemin vers la destination et entraine des messages de contrôle supplémentaires suite à l'initialisation d'un nouveau processus de création de route, ce qui dégrade la qualité de service.

2.1.2.2 Modification d'une réponse de route.

Comme pour les demandes de route, un nœud malhonnête peut **modifier** un ou plusieurs champs qu'il n'est pas supposé modifier avant de retransmettre le message. La modification peut aussi porter sur un champs qu'il a le droit de modifier, mais il ne respecte pas la spécification pour le faire. Le tableau 2.4 présente les champs où le nœud malhonnête peut intervenir.

Le nœud malhonnête peut jouer sur le numéro de séquence de la destination et/ou le nombre de saut dans une RREP en augmentant le premier et en diminuant le second. Ces paramètres sont pris en compte lors de la mise à jour du chemin vers la destination : une mise à jour est possible si le numéro de séquence reçu dans la demande de route est plus grand que celui stocké dans la table de routage ou les numéros de séquence sont égaux et le nombre de sauts reçu est plus petit

que celui stocké dans la table de routage. Cette intervention permet de garder le chemin qui passe par le nœud malhonnête même si un autre chemin plus court est proposé par un autre nœud.

Champs de la RREP	Est modifiable ?	Modification	Effet
Type	Non	Changer le type	Rejet du message (non conforme au type déclaré)
Nombre de sauts	Oui (autorisé +1)	Diminuer	Fausser le nombre de sauts vers la destination
		Augmentation (>1)	
Adresse destination	Non	Changer	Rediriger le paquet (trafic inutile)
Adresse source	Non	Changer	
Numéro de séquence Destination	Non	Augmenter	Forcer à garder le chemin avec le plus grand numéro de séquence en cas de RREP multiples
		Diminuer	Diminuer les chances en cas de RREP multiples
Drapeau A	Non	inverser (0|1)	Nécessité d'un accusé si égal à 1

TABLE 2.4 – Modifications possibles sur les champs des RREP

2.1.2.3 Fabrication d'une réponse de route.

Certaines attaques peuvent être effectuées sans pour autant être choisi sur le chemin. C'est le cas des attaques par **fabrication** :

- Fausse réponse : à la réception d'une demande de route, le nœud malhonnête fabrique une réponse de route même s'il n'a pas de chemin valide vers la destination. Dans un autre cas de figure, le nœud malhonnête répond avec une réponse de route même s'il n'est pas supposé le faire lorsque le drapeau D est à 1 (indiquant que seulement la destination doit répondre).
- Réponse active : des réponses de routes sont fabriquées et injectées dans le réseau même sans avoir reçu une demande de route au préalable. Dans ce cas le nœud malveillant peut jouer sur tout les champs précédemment présentés pour produire l'effet désiré. Une variante de cette attaque vise à déborder la table de routage d'une cible en proposant des routes (via RREP) vers des nœuds (nouveaux ou inexistants).
- En écoutant la transmission d'une réponse de route qui ne lui est pas destinée, un nœud malhonnête peut fabriquer et injecter un paquet RREP proposant un chemin plus court et plus frais provoquant la mise à jour du chemin vers la destination qui passe dorénavant par le nœud malveillant.

2.1.3 Attaques élémentaires portant sur les erreurs de route

2.1.3.1 Suppression d'une erreur de route

Comme c'est le cas pour les RREQ et RREP, en **effaçant** une RERR, un nœud malhonnête peut retarder la détection des liens défaillants. Cependant, l'impact de cette attaque est restreint du fait que les nœuds en amont découvrent le problème et demandent l'établissement de nouvelles routes.

2.1.3.2 Modification d'une erreur de route

Un nœud malhonnête peut **modifier** des erreurs de route avant de les retransmettre. Le tableau 2.5 indique les champs où le nœud malveillant peut intervenir. Ainsi, il peut supprimer des

destinations non-joignables pour faire croire qu'elles le sont encore et ajouter des destinations qui sont joignables et actives pour faire croire qu'elles ne le sont plus et les désactiver.

Champs de la RERR	Est modifiable ?	Modification	Effet
Type	Non	Changer le type	Rejet du message (non conforme au type déclaré)
Nombre de destinations	Non	Incrémenter	Ajouter une destination non joignable
		Diminuer	Supprimer une destination non-joignable
Adresse destination non joignable	Non	Changer	Faire croire qu'une autre destination est non joignable
Numéro de séquence destination non joignable	Non	Augmenter	Mise à jour des entrée des voisins correspondant à cette destination
		Diminuer	Rendre l'entrée correspondante non valide
Ajouter autant de destinations à déclarer comme non-joignables et leur numéro de séquence			

TABLE 2.5 – Modifications possibles sur les champs des RERR

2.1.3.3 Fabrication d'une erreur de route

Un nœud malhonnête peut **fabriquer** un message d'erreur de route et déclarer autant de routes non-joignables causant l'invalidation des entrées correspondantes dans la table de routage des nœuds recevant le message de contrôle.

2.1.4 Attaques composées

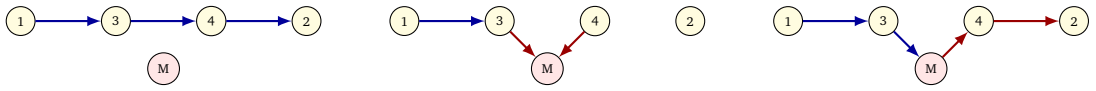
Comme nous l'avons indiqué plus haut, un attaquant peut combiner des attaques élémentaires pour effectuer des attaques composées potentiellement pour atteindre des objectifs plus évolués. Nous présentons dans ce qui suit des exemples d'attaques composées exploitant les vulnérabilités du protocole de routage ad hoc AODV.

2.1.4.1 Répétition régulière d'attaques élémentaires

Cette attaque se base sur une répétition régulière d'une attaque élémentaire pour avoir un impact permanent. Par exemple, l'envoi continue de messages de demande de route RREQ fabriqués à destination d'une cible est efficace pour empêcher celle-ci de recevoir les messages des autres nœuds. De même, l'envoi de réponses de route fabriquées est efficace pour empêcher la cible d'envoyer des messages aux autres nœuds (puisque le nœud malhonnête devient le prochain saut vers les autres nœuds). En combinant ces deux attaques composées, un nœud malhonnête peut isoler sa cible.

2.1.4.2 Insertion dans une route déjà établie

Une attaque composée est présentée dans [NS03] qui se base sur l'utilisation de deux demandes de route fabriquées pour s'insérer sur une route déjà établie entre une source et une destination. Dans cette attaque, le nœud malicieux compte sur sa proximité d'au moins deux nœuds faisant partie de la route sur laquelle s'insérer. Nous expliquons le déroulement de cette attaque par l'exemple de la figure 2.1.



a. Initialement le chemin 1-3-4-2 est établie et M peut entendre les transmissions de 3 et 4

b. Après transmission de la première RREQ fabriquée

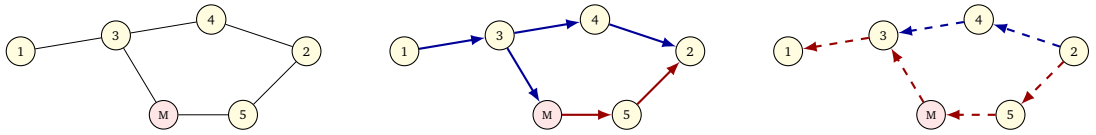
c. Après transmission de la deuxième RREQ fabriquée

FIGURE 2.1 – Invasion de route (route déjà existante)

La figure 2.1a montre l'état initial du réseau où 1 et 2 ont établi une connexion les reliant via les nœuds 3 et 4. Le nœud M, à proximité de 3 et 4, tente de s'introduire sur la route : il fabrique une première demande de route en initialisant les champs comme suit : (1) l'adresse source est initialisée à 2, (2) l'adresse destination est initialisé à 1, et (3) le numéro de séquence de la source 2 est fixé à une valeur supérieure à la valeur actuelle. Il suffit enfin de fixer l'adresse IP de la source dans l'entête IP à l'adresse du nœud malhonnête (M) avant de diffuser le message. Les nœuds 3 et 4 recevant la demande de route vont choisir M comme prochain saut en direction de la source 2 dans leurs tables de routage (voir figure 2.1b). Ensuite, le nœud malhonnête fabrique une seconde demande de route pour rétablir le lien vers 2. Cette RREQ fabriquée contient dans : (1) l'adresse source = M, (2) l'adresse destination = 2, et (3) le numéro de séquence de la destination est initialisé à une valeur plus grande que la valeur actuelle. En diffusant cette demande de route, il obtient une route vers la destination (voir figure 2.1c).

2.1.4.3 Insertion dans une route non encore établie

Une autre attaque composée existe pour s'insérer sur un chemin. Celle-ci s'opère lors du processus de découverte de route (route non encore établie). Elle nécessite l'utilisation d'un paquet de demande de route et un paquet de réponse de route. Nous expliquons cette attaque à travers l'exemple de la figure 2.2. Il est à noter que pour cette attaque, le nœud malhonnête n'est pas obligé d'être à proximité des nœuds qui seront choisis dans le chemin.



a. Réseau avant l'initialisation de la découverte du chemin où les traits entre les nœuds représente la visibilité de chaqu'un d'eux

b. Propagation de la RREQ de initialisée par 1 (en bleu) et celle initialisée par M (en rouge)

c. Propagation des réponses de route correspondant à RREQ_1 en bleu et RREQ_M en rouge

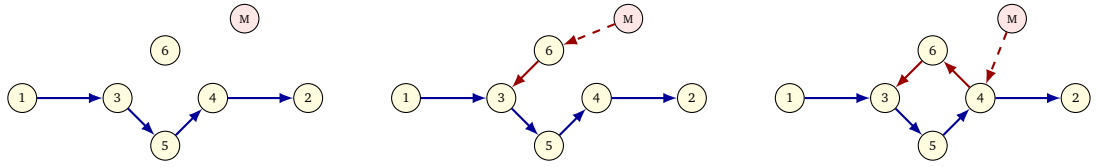
FIGURE 2.2 – Invasion de route (lors de l'établissement du chemin)

À la réception de la demande de route initialisée par le nœud 1, l'attaquant M créé une demande de route normale comme si lui aussi voulait établir un chemin vers la même destination demandée dans la RREQ qu'il vient de recevoir (à savoir 2). Cette action garantie qu'il obtient un chemin vers la destination quelque soit le chemin qui sera retenu pour la demande de 1. Ensuite, à la réception de la réponse de route, il fabrique une réponse de route à destination de 1 où il augmente le numéro de séquence de la destination qu'il vient de recevoir et transmet le paquet vers le prochain saut en direction de la source 1 (vers le nœud 3). Ceci assure que le chemin vers la destination

proposé par la réponse de route fabriquée est choisie puisqu'il est plus frais. De cette manière, l'attaquant s'insère entre la source et la destination.

2.1.4.4 Création d'une boucle de routage

Ning et Sun présentent dans [NS03] une attaque permettant la formation d'une boucle de routage dans une route déjà établie en fabriquant deux réponses de routes. Nous présentons cette attaque à travers l'exemple de la figure 2.3a où on suppose l'existence d'un chemin entre le nœud 1 et le nœud 2 passant par 3, 5 et 4. Le nœud malhonnête M fabrique une première réponse de route où l'adresse source est initialisée à 1, l'adresse destination à 2 et un numéro de séquence de la destination 2 supérieur au numéro de séquence actuel. Il fait croire que ce paquet est envoyé par le nœud 3 à destination du nœud 6 (valeurs à mettre respectivement dans le champs adresse source et destination de l'entête IP). Ceci provoque l'envoi de tout paquet reçu à destination de 2 vers le nœud 3 (voir figure 2.3b). Ensuite, M fabrique une deuxième réponse de route équivalente à la première sauf qu'il fait croire qu'elle est à destination du nœud 4, provenant du nœud 6. Cette action garantie que tout paquet transféré vers 2 à travers 4 est envoyé vers 6, ce qui complète la boucle.



a. Initialement le chemin entre 1 et 2 passe par 3-5-4 respectivement

b. Après transmission de la première RREP fabriquée

c. Après transmission de la deuxième RREP fabriquée, la boucle est formée

FIGURE 2.3 – Création d'une boucle de routage dans une route déjà existante

2.1.4.5 Création d'un tunnel

L'attaque du trou de ver peut aussi être facilement appliquée sur le protocole de routage AODV. Elle a été présentée dans [TBK+03] comme étant une attaque agrégée. Elle nécessite la coopération de deux ou plusieurs nœuds malhonnêtes qui falsifient la longueur des routes. Ils proposent des routes plus courtes grâce au tunnel qu'ils construisent. Ensuite, le nœud malicieux recevant un message l'encapsule pour le transférer vers l'autre bout du tunnel où il sera décapsulé et rediffusé. De cette manière, les différents sauts formant le tunnel ne sont pas comptabilisés et ainsi le chemin obtenu parait plus court.

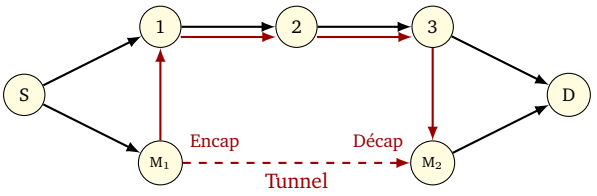


FIGURE 2.4 – Attaque du tunnel

La figure 2.4 montre un exemple de tunnel créé entre les nœuds malicieux M_1 et M_2. D reçoit deux chemins : [S–1–2–3–D] et [S–M_1–M_2–D]. Le chemin le plus court est choisi (à savoir [S–M_1–M_2–D]). Or ce chemin est en réalité plus long [S–M_1–1–2–3–M_2–D].

Après avoir passé en revue les attaques qu'un nœud malhonnête peut faire, nous affirmons que ces attaques se basent essentiellement sur des manipulations élémentaires des messages de routage échangées. Elles exploitent la confiance que les nœuds se vouent pour propager leurs actions malveillantes. Ceci nous amène à analyser le comportement normal du protocole pour trouver les moyens qui permettrons de détecter ces actions. Nous utilisons dans cette analyse un langage ([YKB93]) permettant la formalisation du comportement normal explicitant les aspects de confiance qui existent dans AODV sous forme de règles. Ensuite, nous montrons que nous pouvons enrichir la connaissance de chaque nœud afin de détecter les comportements incorrects, c'est-à-dire ceux qui violent les règles de confiance.

2.2 Analyse de la confiance implicite dans AODV

2.2.1 Présentation du langage de formalisation

Nous utilisons le langage proposé par Yahalom et al. [YKB93] pour la modélisation des règles de confiance. La notion de confiance sous-jacente à ce langage est exprimé par le fait qu'une entité P fait confiance à une entité Q à propos de certains aspects. Cela signifie que P croit que Q se comportera d'une certaine façon, réalisant (ou non) une certaine action dans des circonstances spécifiques. Ces relations de confiance sont de deux types : directes et dérivées. Les relations de confiance dérivées sont obtenues à partir des recommandations d'autres entités.

Dans le contexte de l'analyse des protocoles d'authentification, [YKB93] définit 7 classes de confiance en fonction des actions considérées et des circonstances d'exécution. Nous citons à titre indicatif : (i) la classe *ID* pour l'identification d'entités, (ii) la classe *KG* pour la génération aléatoire de clé, (iii) la classe *KS* pour le maintient du secret, et (iv) la classe *PS* pour la réalisation correcte des étapes du protocole. Mais, nous ne pouvons pas utiliser ces classes dans le contexte des protocoles de routage ad hoc. C'est pourquoi nous définissons de nouvelles classes qui décrivent effectivement les actions effectuées par le protocole. Nous les détaillons plus tard dans cette section.

[YKB93] propose les notations suivantes pour exprimer les clauses relatives à la confiance :

- $P\ trusts_{cc}\ (Q)$: P a confiance en Q pour réaliser l'action cc.
- $P\ trusts_{cc}\ (S)$: P a confiance en toutes les entités de S (ensemble d'entités qui vérifient des conditions) pour réaliser l'action cc.
- $P\ \neg trusts_{cc}\ (Q)$: P ne fait pas confiance à Q pour faire l'action cc.

2.2.2 Notations

Nous utilisons les notations suivantes pour représenter les informations sur la topologie du protocole AODV :

- NS_X L'ensemble des voisins connus à un saut du nœud X. Il contient les nœuds détectés dans le voisinage de X. C'est en fait un sous ensemble de la table de routage contenant les voisins à un saut.
- HT_X Table d'historique du nœud X contenant les demandes de route RREQ traitées. Ceci revient à garder trace de l'identifiant de la requête $\#ID$ et de l'adresse de la source $@Src$ de la RREQ. La section B.1.2 page 116 présente cette structure avec plus de détails.

  Nous définissons deux fonctions pour la manipulation de cette structure :

  - $add_{HT}(RREQ, HT)$ permet le stockage de la demande de route traité dans la table d'historique (HT). Il s'agit de créer une nouvelle entrée dans cette table contenant l'identifiant de la demande de route $\#ID$ et l'adresse IP de la source $@Src$ en leurs associant une durée de vie (*LifeTime*).
  - $isTreated(RREQ, HT)$ permet de vérifier si la demande de route reçue a été déjà traité ou non.
- RT_X Table de routage du nœud X. La structure de cette table est présentée dans l'annexe B.1.1 (page 115). Chaque entrée contient entre autres les informations suivantes : [$@D$, $\#SN$, $\#HC$, $@NH$, PL, LT] respectivement adresse IP de la destination, son numéro de séquence, le nombre de saut pour l'atteindre, le prochain saut en sa direction, la liste des précurseurs et la durée de vie de l'entrée.
- $RT_M.X.Y$ est utilisé pour accéder à n'importe quelle information de la table de routage : X est la ligne (chaque ligne est identifiée par $@D$) et Y est la colonne (peut être $\#SN$, $\#HC$, $@NH$, PL ou LT) de la table de routage du nœud M. Par exemple, $RT_M.N.SN$ représente le numéro de séquence du nœud N dans la table de routage du nœud M.
- $add_{RT}((@D, \#SN, \#HC, @NH, PL, LT), RT)$ est une fonction permettant d'ajouter l'entrée ($@D$, $\#SN$, $\#HC$, $@NH$, PL, LT) à la table de routage (RT).
- $update((@D, \#SN, \#HC, @NH, PL, LT), RT)$ est une fonction permettant de mettre à jour une entrée spécifique ($@D$, $\#SN$, $\#HC$, $@NH$, PL, LT) dans la table de routage RT.

Les messages échangés lors du processus de découvertes de route sont représentés comme suit :

- [$\#ID$, $@D$, $\#SND$, $@Src$, $\#SNS$, $\#HC$] : représente un paquet de demande de route $RREQ$ contenant dans l'ordre l'identifiant de la RREQ, l'adresse de la destination et son numéro de séquence, l'adresse de la source et son numéro de séquence et le nombre de sauts pour atteindre la source. Pour plus de détails concernant la structure et la disposition des champs du paquet de demande de route, veuillez vous référer à la section B.2.1 à la page 116.
- [$@D$, $\#SND$, $@Src$, $\#HC$] : représente un paquet de réponse de route $RREP$. Ce paquet contient l'adresse de la destination ainsi que son numéro de séquence, l'adresse de la source et le nombre de sauts pour atteindre la destination. La section B.2.2, page 117, présente avec plus de détails la structure et la disposition des champs de ce paquet.

L'échange de messages est représenté comme suit :

- $MSG_X.Y$ est un champs particulier (Y) du message MSG envoyé par le nœud X. MSG est l'un des messages de contrôle d'AODV (RREQ, RREP, HELLO ou RERR). Par exemple, $RREQ_A.@D$ est l'adresse IP de la destination que le nœud A cherche à joindre en initialisant

une découverte de route RREQ.

- $X \xrightarrow{\text{MSG}_X} *$: Le nœud X diffuse le message MSG.
- $X \xleftarrow{\text{MSG}_Y}$: Le nœud X reçoit le message (MSG) diffusé par Y.
- $X \xrightarrow{\text{MSG}_X} Y$: Le nœud X envoi en unicast le message (MSG) au nœud Y.
- $X \xleftarrow{\text{MSG}_Y} Y$: Le nœud X reçoit le message (MSG) envoyé en unicast par le nœud Y.

Dans ce qui suit, nous nous focalisons sur le comportement d'un nœud exécutant le protocole AODV. Nous présentons les actions faites en interne à chaque étape du processus de découverte et de maintient de route. Ensuite, nous récapitulons ces actions sous forme de règles de confiance puisque ce processus n'aboutit que si les nœuds coopèrent et croient en l'exactitude des informations fournies. En d'autres termes, un nœud fait confiance aux voisins sur le fait qu'ils se comportent normalement. Cette règle a la forme suivante :

Règle 2.x

$$X\ trusts_{cc}\ (Y)\ :$$
$$Conditions\ observées\ selon\ les\ messages\ reçus$$
$$\Rightarrow\ \ Traitement\ attendu\ en\ conséquence$$

Il est à noter que ces règles se composent de deux parties. Dans la première ($X\ trusts_{cc}\ (Y)$), nous annonçons une relation de confiance du nœud envers un autre à propos d'un ensemble d'actions. Dans la deuxième partie, nous présentons le traitement attendu lorsque les conditions nécessaires sont réunies. Cette partie est sous la forme d'une implication.

2.2.3 Processus de découverte de route

2.2.3.1 Initialisation de la demande de route

Avant d'envoyer un message à la destination, le nœud source vérifie l'existence d'une route valide dans sa table de routage. Ainsi, s'il ne connait pas la destination ou s'il possède un chemin obsolète, le nœud doit initier une découverte de route. Pour ce faire, il diffuse une demande de route RREQ.

Une source peut imposer que seulement la destination lui réponde en mettant le drapeau *D* (*destination_only*) à 1 dans la demande de route. Ce bit impose à tous les nœuds intermédiaires de rediffuser la demande de route vers la destination et de ne pas répondre avec une RREP même s'ils ont déjà un chemin valide. Dans le cas contraire (bit *destination_only* à zéro), un nœud intermédiaire a le droit de répondre. Nous supposons que la valeur de ce drapeau est celle par défaut à savoir zéro c'est-à-dire qu'un nœud intermédiaire a le droit de répondre s'il a un chemin valide vers la destination.

Une source peut imposer le fait que la liaison établie avec la destination soit symétrique. Il s'agit dans ce cas de mettre le bit *G* à 1 (*Gratuitous RREP*). Ce cas se présente lorsque le bit *destination_only* est à zéro et les nœuds intermédiaires ayant un chemin vers la destination répondent

par des RREP. Ceci implique que la destination ne reçoit pas la demande de route et ainsi n'établit pas de chemin vers la source. Avec le drapeau *G* à 1, la source impose aux nœuds intermédiaires qui peuvent répondre par une RREP, de créer une RREP supplémentaire (*Gratuitous RREP*) qui est envoyée en direction de la destination lui indiquant que telle source a cherché à la joindre. De cette manière, la destination a aussi un chemin vers la source et la liaison obtenue à la fin du processus de découverte de route est symétrique. Comme pour le bit *D*, nous supposons que le bit *G* est à zéro (valeur par défaut).

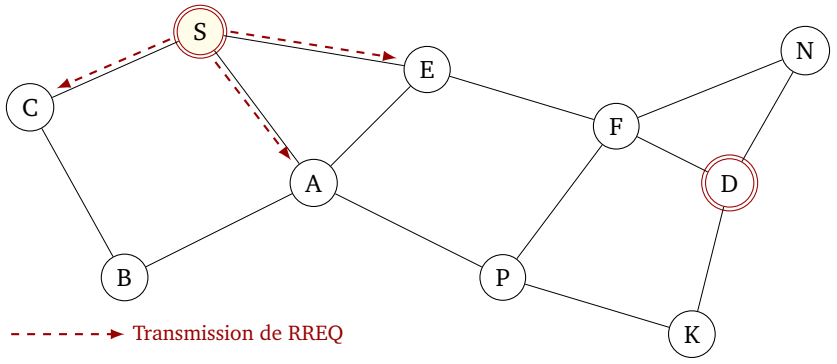


FIGURE 2.5 – Initialisation de la demande de route par *S*

Dans la figure 2.5, le nœud source *S* essaie d'établir une route vers la destination *D*. Ainsi, après avoir cherché une route valide dans sa table de routage et ne pas l'avoir trouvé, *S* initialise une demande de route ($RREQ_S$). Pour la création de cette requête, *S* commence par incrémenter son numéro de séquence ($\#SNS = \#SNS+1$) ainsi que son identifiant de requêtes ($\#ID = \#ID+1$) pour éviter les conflits avec les demandes de routes précédemment diffusées. Ensuite, *S* ajoute ces valeurs à la demande de route qu'il prépare. Cette demande de route aura la forme suivante :

$$RREQ_S \; = \; [\, \#ID,\; @D = D,\; \#SND,\; @Src = S,\; \#SNS,\; \#HC = 0\,]$$

Il ne reste plus à *S* qu'à ajouter cette demande de route à sa table d'historique (add_{HT}) et enfin diffuser le message. Nous rappelons que la table d'historique sert à vérifier si la demande de route fraichement reçue n'est pas déjà traitée.

L'expression 2.1 résume les actions faites par un nœud pour l'initialisation d'une demande de route.

Expression 2.1

$$\#ID = \#ID + 1 \;\; \wedge \;\; \#SNS = \#SNS + 1 \;\; \wedge \;\; add_{HT}(RREQ_S, HT_S) \;\; \wedge \;\; S \xrightarrow{RREQ_S} *$$

2.2.3.2 Propagation de la demande de route

Chaque nœud intermédiaire recevant la demande de route la retransmet après avoir effectué des mises à jour concernant le paquet RREQ ainsi que ses structures internes. Ce traitement est fait une seule fois pour une requête donnée même si cette demande de route peut être reçue plusieurs fois de différents voisins.

Lors de la réception d'une demande de route $RREQ_N$ par le nœud M,

$$M \xleftarrow{\text{RREQ}_N} \tag{1}$$

il effectue les actions suivantes :

- *M* crée ou met à jour un chemin vers le nœud duquel le message est reçu selon que ce nœud appartient ou non à sa table de routage ;

$$[\ (\ N \notin NS_M\ \Rightarrow\ add_{RT}(Entry_N, RT_M)\ \vee\ (\ N \in NS_M\ \Rightarrow\ update(Entry_N, RT_M)\ ] \tag{2}$$

  où $Entry_N$ est l'entrée correspondant au voisin *N* duquel le paquet est reçu et contenant [N, ?, 1, N, $\emptyset$, LT] (« ? » signifiant que l'entrée est ajoutée à la table de routage sans numéro de séquence puisque cette information n'est pas reçue dans le paquet RREQ).

- *M* vérifie l'existence de cette demande de route dans la table d'historique. Il parcourt pour ce faire cette structure à la recherche du couple $[RREQ_N.\#ID,\ RREQ_N.@Src]$. Si cette demande de route est trouvée,

$$isTreated(RREQ_N,\ HT_M) =\ True \tag{3}$$

  le nœud traitant supprime cette demande et arrête le traitement.

  Cependant, si cette demande n'est pas trouvée,

$$isTreated(RREQ_N,\ HT_M) =\ False \tag{4}$$

  M ajoute ou met à jour un chemin vers la source dans sa table de routage ;

$$\begin{aligned}[\ (\ RREQ_N.@Src \notin RT_M\ \Rightarrow\ add_{RT}(Entry_{RREQ_N.@Src}, RT_M)\quad\ \vee\\ (\ RREQ_N.@Src \in RT_M\ \Rightarrow\ update(Entry_{RREQ_N.@Src}, RT_M)\ ]\end{aligned} \tag{5}$$

  où $Entry_{RREQ_N.@Src}$ est l'entrée correspondant à la source ($@Src$) à ajouter ou à mettre à jour dans la table de routage ($[RREQ_N.@Src,\ RREQ_N.\#SNS,\ RREQ_N.\#HC + 1,\ N,\ \emptyset,\ LT]$).

  M ajoute la demande de route à la table d'historique ;

$$add_{HT}(RREQ_N, HT_M) \tag{6}$$

- Enfin, *M* vérifie l'existence d'un chemin valide vers la destination dans sa table de routage. Son action dépend aussi de son positionnement : s'agit-il d'un nœud intermédiaire ou du nœud destination ?
  - Si *M* n'est pas la destination ($M \neq RREQ_N.@D$) et n'a pas de chemin valide vers la destination, il rediffuse la demande de route modifiée. La modification de la demande de route porte sur le champs nombre de saut qui est incrémenté de un pour compter le saut vers le nœud traitant (*M*).

$$RREQ_M.\#HC\ =\ RREQ_N.\#HC\ +\ 1$$

De plus, *M* garde dans la RREQ le plus grand numéro de séquence destination ($\#SND$) entre celui reçu dans le message RREQ ($RREQ_N.\#SND$) et celui gardé dans sa table de routage pour l'entrée destination si cette entrée existe ($RT_M.(RREQ_N.@D).\#SN$).

$$RREQ_M.\#SND = MAX(RREQ_N.\#SND,\ RT_M.(RREQ_N.@D).\#SN)$$

Ceci permet de faire parvenir le plus grand numéro de séquence de destination ($\#SND$) connus par les nœuds intermédiaires relayant la RREQ à la destination. Celle-ci se charge de renvoyer une réponse de route fraiche au yeux de tous les nœuds puisque le numéro de séquence qu'elle va garder est le plus grand entre celui reçu et son numéro de séquence actuel. Il est à noter que le nœud traitant *M* ne doit pas modifier la valeur stockée dans sa table de routage même si la valeur qui est dans la RREQ est supérieure.

Ce qui revient à :

$$\begin{array}{c}[\ RREQ_N.@D \notin RT_M\ \vee\ (RREQ_N.@D \in RT_M\ \wedge\ RREQ_N.@D\ invalid)\ ] \\ \wedge\ M \neq RREQ_N.@D\ \ \Rightarrow\ \ M \xrightarrow{\text{RREQ}_M} *\end{array} \quad (7)$$

– Lorsque *M* n'est pas la destination ($M \neq RREQ_N.@D$) mais qu'il a un chemin valide et frais vers la destination, il peut créer une réponse de route qu'il renvoie à la source [1]. Le nœud *M* copie l'adresse source et destination de la demande de route vers les champs correspondants dans la réponse de route. Il copie ensuite la valeur du numéro de séquence qu'il a dans sa table de routage pour la destination $RT_M.(RREQ_N.@D).\#SN$ dans le champs correspondant de la RREP et initialise le nombre de sauts à la valeur du champs nombre de sauts de l'entrée destination de la table de routage $RT_M.(RREQ_N.@D).\#HC$. La réponse de route créée est la suivante :

$$[RREQ_N.@D,\ RT_M.(RREQ_N.@D).\#SN,\ RREQ_N.@Src,\ RT_M.(RREQ_N.@D).\#HC]$$

De plus, le nœud intermédiaire M effectue deux mises à jour au niveau de sa table de routage : la première concerne la liste des précurseurs de l'entrée destination à laquelle est ajoutée le nœud duquel la RREQ est reçue :

$$RT_M.(RREQ_N.@D).PL\ =\ RT_M.(RREQ_N.@D).PL\ \cup\ \{N\}$$

La seconde concerne la liste des précurseurs de l'entrée source à laquelle est ajoutée le prochain saut en direction de la destination :

$$RT_M.(RREQ_N.@Src).PL = RT_M.(RREQ_N.@SRC).PL \cup \{RT_M.(RREQ_N.@D).@NH\}$$

Ainsi,

$$\begin{array}{c}M \neq RREQ_N.@D\ \ \wedge\ \ (RREQ_N.@D \in RT_M\ \wedge\ RREQ_N.@D\ valid) \\ \Rightarrow\ \ update(Entry_{RREQ_N.@Src}, RT_M)\ \ \wedge\ \ update(Entry_{RREQ_N.@D}, RT_M) \\ \wedge\ M\ \xrightarrow{\text{RREP}_M}\ RT_M.(RREP_N.@Src).@NH\end{array} \quad (8)$$

1. Ce traitement est possible lorsque le bit `destination_only` est à zéro. Nous rappelons que nous supposons que ce bit est à zéro.

– Lorsqu'il s'agit de la destination, elle copie l'adresse de la source et de la destination de la demande de route reçue ($RREQ_N.@Src$ et $RREQ_N.@D$) dans les champs correspondant de la réponse de route. Ensuite, elle place son numéro de séquence qui est préalablement mis à jour. Ainsi, le numéro de séquence local est incrémenté de un si la valeur reçue dans la RREQ est égale à cette valeur incrémentée.

$$(\#SND + 1 = RREQ_N.\#SND) \Rightarrow (\#SND = RREQ_N.\#SND)$$

Enfin, D initialise le nombre de sauts à zéro. Ainsi, la réponse de route obtenue est :

$$[RREQ_N.@D,\ \#SND,\ RREQ_N.@Src,\ 0]$$

Ceci correspond à :

$$M = RREQ_N.@D \Rightarrow D \xrightarrow{\text{RREP}_D} RT_D.(RREP_N.@Src).@NH \quad (9)$$

Chaque nœud fait confiance aux voisins entendant sa demande de route, pour la rediffuser en respectant le comportement décrit ci-dessus jusqu'à ce qu'elle atteigne la destination. Ce traitement est repris dans la règle 2.1. Nous présentons dans un premier temps l'association logique entre les différentes expressions en utilisant leurs numéros respectifs. Dans un second temps, nous remplaçons chaque expression par son contenue. Cette règle met en pratique la première classe de confiance *FwRREQ* (*forward route request*) décrivant le comportement normal lors de la réception d'une demande de route.

Règle 2.1

$$N\,trusts_{FwRREQ}\,(M)\ :\ (1) \wedge (4) \Rightarrow (2) \wedge (5) \wedge (6) \wedge ((7) \vee (8) \vee (9))$$

$$N\,trusts_{FwRREQ}\,(M)\ :$$
$$M \xleftarrow{RREQ_N} \wedge\ isTreated(RREQ_N, HT_M) = False$$

$$\Rightarrow [\,(N \notin NS_M \Rightarrow add_{RT}(Entry_N, RT_M)) \vee (N \in NS_M \Rightarrow update(Entry_N, RT_M))\,]$$

$$\wedge \left[\begin{array}{r} (RREQ_N.@Src \notin RT_M \Rightarrow add_{RT}(Entry_{RREQ_N.@Src}, RT_M)) \\ \vee\ (RREQ_N.@Src \in RT_M \Rightarrow update(Entry_{RREQ_N.@Src}, RT_M)) \end{array}\right]$$

$$\wedge\ add_{HT}(RREQ_N, HT_M)$$

$$\wedge \left[\begin{array}{c} \left(\begin{array}{c} [\,RREQ_N.@D \notin RT_M \vee (RREQ_N.@D \in RT_M \wedge RREQ_N.@D\ invalid)\,] \\ M \neq RREQ_N.@D \wedge \Rightarrow M \xrightarrow{RREQ_M} * \end{array}\right) \\ \vee \\ \left(\begin{array}{c} M \neq RREQ_N.@D \wedge (RREQ_N.@D \in RT_M \wedge RREQ_N.@D\ valid) \\ \Rightarrow update(Entry_{RREQ_N.@Src}, RT_M) \wedge update(Entry_{RREQ_N.@D}, RT_M) \\ \wedge\ M \xrightarrow{RREP_M} RT_M.(RREP_N.@Src).@NH \end{array}\right) \\ \vee \\ \left(\ M = RREQ_N.@D \Rightarrow D \xrightarrow{RREP_D} RT_D.(RREP_N.@Src).@NH\ \right) \end{array}\right]$$

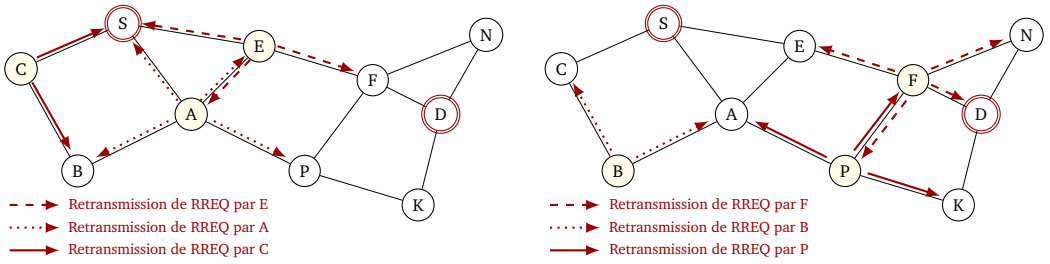
La règle 2.2 décrit le comportement normal lorsque M a déjà traité la demande de route :

Règle 2.2

$$N\,trusts_{\neg FwRREQ}\,(M)\ :\ (1)\ \wedge\ (3)\ \Rightarrow\ (2)$$

$$N\,trusts_{\neg FwRREQ}\,(M)\ :$$
$$M \xleftarrow{RREQ_N}\ \wedge\ isTreated(RREQ_N, HT_M) = True$$
$$\Rightarrow\ [\,(N \notin NS_M\ \Rightarrow\ add_{RT}(Entry_N, RT_M))\ \vee\ (N \in NS_M\ \Rightarrow\ update(Entry_N, RT_M))\,]$$

L'exemple de la figure 2.6 montre la propagation de la demande de route RREQ jusqu'à atteindre la destination *D*. Certains nœuds du réseau ont reçu la même demande de route RREQ à maintes reprises. C'est le cas par exemple, du nœud *A* qui a reçu la même RREQ des nœuds *S*, *E*, *B* et *P*. Il n'a cependant retransmis le paquet qu'une seule fois (figure 2.6a) suite à la réception de la première RREQ qui lui est parvenu du nœud *S* (figure 2.5).



a. Retransmission des nœuds à 1 saut (A, C et E) b. Retransmission par les nœuds à 2 sauts (B, F et P)

FIGURE 2.6 – Retransmission des RREQ par les nœuds intermédiaires

Étape n°	1	2			3			4	
RT	$\#HC{=}0$	$\#HC{=}1$			$\#HC{=}2$			$\#HC{=}3$	
	Trx S	Trx C	Trx E	Trx A	Trx F	Trx B	Trx P	Trx K	Trx N
S		C	E	A					
A	S		E			B	P		
B		C,S		A					
C	S					B			
E	S			A	F				
F			E,S				P		N
K							P,S		
P				A,S	F			K	
N					F,S				
D					F,S			K	N

* Trx M : Transmission du nœud M

TABLE 2.6 – Changements des tables de routage après retransmission de la RREQ

La table 2.6 montre l'évolution des tables de routage (initialement vides) au fur et à mesure de la propagation de la demande de route dans le réseau. Plus exactement, on représente les entrées (routes) qui sont ajoutées dans la table de routage de chaque nœud recevant et traitant une RREQ. Ainsi, Chaque colonne correspond à une retransmission de la RREQ. Ces demandes de routes qui transitent dans le réseau diffèrent essentiellement par le nombre de sauts ($\#HC$) et par l'adresse

du nœud les retransmettant.

Ainsi, les lignes du tableau 2.6 contiennent les entrées que chaque nœud a ajouté dans sa table de routage à la fin de la première partie du processus de découverte de route : la propagation des demandes de route qui se termine par la génération d'une ou plusieurs réponses de route. Dans ce qui suit, nous décrivons la propagation de la réponse de route jusqu'à ce qu'elle atteigne la source.

2.2.3.3 Propagation de la réponse de route

Chaque nœud intermédiaire *M* recevant une réponse de route qui le concerne (l'adresse de destination (IP_Dest) dans l'entête IP est son adresse)

$$M \xleftarrow{\text{RREP}_N} N \tag{10}$$

doit la transférer au prochain saut en direction de la source après avoir incrémenté le nombre de sauts ($RREP_N.\#HC + 1$). Le prochain saut peut être trouvé en consultant sa table de routage RT_M : il s'agit du champs $@NH$ de l'entrée $RREP_N.@Src$ qu'il a construit lors du passage de la demande de route ($RT_M.(RREP_N.@Src).@NH$).

Il est à noter qu'un nœud ne traite pas un paquet de réponse de route qui ne lui est pas destiné car ce message est filtré par la couche liaison.

Le passage de la réponse de route donne aussi lieu à des mises à jour de la table de routage en utilisant les informations contenues dans le paquet :

– *M* met à jour l'entrée correspondante à l'émetteur du message (IP_Src) dans sa table de routage ;

$$update(Entry_N, RT_M) \tag{11}$$

– Si *M* ne trouve pas d'entrée pour la destination,

$$RREP_N.@D \notin RT_M \tag{12}$$

il l'ajoute à sa table de routage en utilisant les informations reçues dans le paquet.

$$add_{RT}(Entry_{RREP_N.@D}, RT_M) \tag{13}$$

où $Entry_{RREP_N.@D}$ est l'entrée correspondant à la destination ($@D$) à ajouter dans la table de routage. Celle-ci est formée par $[RREP_N.@D,\ RREP_N.\#SND,\ RREP_N.\#HC + 1,\ N,\ RT_M.(RREP_N.@Src).@NH,\ RREP_N.LT]$.

– Lorsque *M* a déjà une entrée pour la destination dans sa table de routage,

$$RREP_N.@D \in RT_M \tag{14}$$

il ne la met à jour avec les nouvelles informations fraîchement reçues

$$update(Entry_{RREP_N.@D}, RT_M) \tag{15}$$

que dans l'un des cas suivants :

– Il n'a pas de numéro de séquence valide pour l'entrée destination dans sa table de routage ;

$$RT_M.(RREP_N.@D).\#SN = ? \tag{16}$$

– Il ne garde que le chemin avec le plus grand numéro de séquence entre celui qui est déjà dans sa table de routage et celui reçu dans la réponse de route. Il s'agit en fait du chemin le plus frais ;

$$RREP_N.\#SND > RT_M.(RREP_N.@D).\#SN \tag{17}$$

– En cas d'égalité du numéro de séquence, il ne garde que le chemin avec le plus petit nombre de sauts. Il s'agit du chemin le plus court.

$$\left(\begin{array}{r} RREP_N.\#SND = RT_M.(RREP_N.@D).\#SN \quad \wedge \\ RREP_N.\#HC + 1 < RT_M.(RREP_N.@D).\#HC \end{array} \right) \tag{18}$$

Ceci est utile dans le cas où un nœud reçoit plus d'une réponse de route. Il devra alors prendre une décision qui dépend ainsi en premier lieu de la fraicheur de la route et en second lieu de la longueur de la route.

– *M* étant un nœud intermédiaire ($M \neq RREP_N.@Src$), il recherche dans sa table de routage l'entrée à la source qui, rappelons-le, a été construite lors du passage de la demande de route pour obtenir l'adresse du prochain saut ($RT_M.(RREP_N.@Src).@NH$) vers lequel la réponse de route est acheminée après avoir incrémenté le nombre de sauts de 1.

$$M \neq RREP_N.@Src \;\Rightarrow\; M \xrightarrow{\text{RREP}_M} RT_M.(RREP_N.@Src).@NH \tag{19}$$

Enfin, *M* met à jour les listes des précurseurs de l'entrée destination et du nœud duquel le RREP est reçue en y ajoutant le nœud vers lequel la RREP est acheminée :

$$RT_M.(RREP_N.@D).PL = RT_M.(RREP_N.@D).PL \cup \{RT_M.(RREP_N.@Src).@NH\}$$

$$RT_M.IP\_Src.PL = RT_M.IP\_Src.PL \cup \{RT_M.(RREP_N.@Src).@NH\}$$

Chaque nœud fait confiance au voisin vers lequel la réponse de route est transférée de la rediffuser vers le prochain saut en direction de la source en respectant le comportement décrit ci-dessus. Ce traitement est repris dans les règles 2.3 et 2.4 : la première (règle 2.3) concerne le cas où le nœud traitant ne connait pas de chemin valide vers la destination et la seconde (règle 2.4) traite le cas où il existe un chemin vers la destination dans la table de routage du nœud traitant. Ces règles utilisent une deuxième classe de confiance *FwRREP* (*forward route reply*) décrivant le comportement normal à la réception d'une réponse de route.

Il est à noter que comme c'est le cas pour les demandes de route, nous présentons dans un premier temps l'association logique entre les différentes expressions en utilisant leurs numéros respectifs, ensuite, dans un second temps, nous remplaçons chaque expression par son contenue.

Règle 2.3

$$N\,trusts_{FwRREP}\,(M)\ :\ (10)\ \wedge\ (12)\ \Rightarrow\ (11)\ \wedge\ (13)\ \wedge\ (19)$$

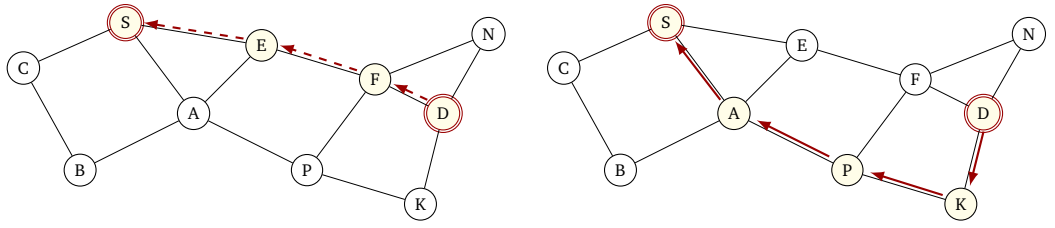
$$N\,trusts_{FwRREP}\,(M)\ :$$
$$M \xleftarrow{RREP_N} N\ \wedge\ RREP_N.@D \notin RT_M$$
$$\Rightarrow\ update(Entry_N, RT_M)\ \wedge\ add_{RT}(Entry_{RREP_N.@D}, RT_M)$$
$$\wedge\ M \neq RREP_N.@Src\ \Rightarrow\ M\ \xrightarrow{RREP_M}\ RT_M.(RREP_N.@Src).@NH$$

Règle 2.4

$$N\,trusts_{FwRREP}\,(M)\ :\ (10)\ \wedge\ (14)\ \wedge\ [\,(16)\ \vee\ (17)\ \vee\ (18)\,]\ \Rightarrow\ (11)\ \wedge\ (15)\ \wedge\ (19)$$

$$N\,trusts_{FwRREP}\,(M)\ :$$
$$M \xleftarrow{RREP_N} N\ \wedge\ RREP_N.@D \in RT_M\ \wedge$$
$$\left[\begin{array}{l}(RT_M.(RREP_N.@D).\#SN\ =?)\ \vee\ (RREP_N.\#SND > RT_M.(RREP_N.@D).\#SN)\\ \quad\vee\left(\begin{array}{r}RREP_N.\#SND = RT_M.(RREP_N.@D).\#SN\ \wedge\\ RREP_N.\#HC + 1 < RT_M.(RREP_N.@D).\#HC\end{array}\right)\end{array}\right]$$
$$\Rightarrow\ update(Entry_N, RT_M)\ \wedge\ update(Entry_{RREP_N.@D}, RT_M)$$
$$\wedge\ M \neq RREP_N.@Src\ \Rightarrow\ M\ \xrightarrow{RREP_M}\ RT_M.(RREP_N.@Src).@NH$$

Pour illustrer ce processus, la figure 2.7a montre le cheminement de la réponse de route initialisée par la destination jusqu'à ce qu'elle atteigne la source. Ainsi, *D* envoie une réponse de route vers *F* qui la transfère à *E* et celui-ci se charge enfin de la délivrer à la source *S*.



a. RREQ transmise par F arrive en premier à D

b. RREQ transmise par K arrive en premier à D

FIGURE 2.7 – Propagation de la réponse de route RREP

Le cas présenté dans cet exemple (figure 2.7a) suppose que la RREQ diffusé par le nœud *F* est la première à atteindre la destination. Le cas serait différent si le nœud *F* était congestionné et la RREQ de *K* atteignait en premier la destination. Dans ce cas (figure 2.7b), le chemin emprunté par la réponse de route serait différent : il passerait par *K*, *P* et *A* avant d'atteindre *S*.

Les tables 2.7a et 2.7b représentent les changements introduits au niveau des tables de routage des nœuds traitant la RREP pour les deux cas de figures 2.7a et 2.7b respectivement.

Nous remarquons que la connaissance des nœuds est enrichie seulement par l'ajout d'un chemin à la destination. De plus, il n'y a que les nœuds relayant la réponse de route qui obtiennent cette connaissance du fait que ce type de messages est transmis en *unicast* (tout autre nœud qui n'est pas la destination du message ne le traite pas et l'efface).

Étape n°		1	2	3
RT	RT initiale	$\#HC{=}0$ Trx D→F	$\#HC{=}1$ Trx F→E	$\#HC{=}2$ Trx E→S
S	C,E,A			D
A	S,E,B,P			
B	C,S,A			
C	S,B			
E	S,A,F		D	
F	E,S,P,N	D		
K	P,S			
P	A,S,F,K			
N	F,S			
D	F,S,K,N			

a. Lorsque la première RREQ arrive par F (cas de la figure 2.7a)

Étape n°		1	2	3	4
RT	RT initiale	$\#HC{=}0$ Trx D→K	$\#HC{=}1$ Trx K→P	$\#HC{=}2$ Trx P→A	$\#HC{=}3$ Trx A→S
S	C,E,A				D
A	S,E,B,P			D	
B	C,S,A				
C	S,B				
E	S,A,F				
F	E,S,P,N				
K	P,S	D			
P	A,S,F,K		D		
N	F,S				
D	F,S,K,N				

b. Lorsque la première RREQ arrive par K (cas de la figure 2.7b)

TABLE 2.7 – Changements des tables de routage après retransmission de la RREP

Ainsi, le chemin est établie et le transfert de données peut débuter. Dans la section suivante, nous présentons comment les nœuds maintiennent la connectivité avec le voisinage.

2.2.4 Maintient des routes

Les messages HELLO sont utilisés pour maintenir les informations de connectivité. À chaque intervalle de temps (*hello_interval*), le nœud vérifie s'il a diffusé un message et si ce n'est pas le cas, il diffuse un message HELLO pour préciser aux voisins qu'il est toujours à porté radio. Il ne s'agit pas d'un message ayant une nouvelle structure : c'est en fait une réponse de route où les champs adresse de destination $@D$ et numéro de séquence $\#SND$ contiennent respectivement sa propre adresse et son numéro de séquence actuel. Ce paquet est destiné au voisinage direct (à un saut) d'où l'initialisation du TTL (*Time-To-Live*) à 1 dans l'entête IP. Il diffère aussi du paquet de réponse de route par le fait qu'il est diffusé (localement) et non envoyé en *unicast*.

À la réception d'un message HELLO, le nœud vérifie s'il a une entrée pour le voisin duquel le message est reçu dans sa table de routage pour la mettre à jour ; sinon il enrichit son voisinage d'un nouveau voisin et ajoute ainsi l'entrée correspondante dans sa table de routage.

Règle 2.5

$$M \xleftarrow{HELLO_N} \begin{cases} & M\,trusts_{Fw}(N)\ : \\ \wedge\, N \notin RT_M \ \Rightarrow\ add_{RT}(Entry_{HELLO_N.@D}, RT_M) \\ \vee \\ \wedge\, N \in RT_M \ \Rightarrow\ update(Entry_{HELLO_N.@D}, RT_M) \end{cases}$$

où $Entry_{HELLO_N.@D} = (HELLO_N.@D,\ HELLO_N.\#SND,\ 1,\ N,\ \emptyset,\ LT)$.

Chaque nœud contrôle aussi la connectivité des voisins. Il vérifie ainsi l'activité de ceux qui sont actifs à chaque intervalle de temps appelé *ACTIVE_ROUTE_TIMEOUT* en cherchant s'ils ont envoyé un message au cours de cet intervalle. Le message peut être une notification de la couche liaison ou un message de contrôle spécifique AODV (RREQ, HELLO, ...). Si la connectivité à un voisin ne peut pas être déterminée, le nœud doit supposer que le lien est perdu et prendre les mesures correctives nécessaires :

- Invalider les routes concernées par la perte de ce lien partant du fait que si un nœud (voisin direct) n'est plus joignable, tous les chemins passant par ce voisin ne sont plus joignables. Le nœud détectant la rupture d'un lien commence par marquer comme invalide le chemin vers le voisin direct dans sa table de routage ensuite tous les chemins l'utilisant comme prochain saut ;
- Créer une liste des destinations affectées contenant l'ensemble des nœuds qui ne sont plus accessibles après la rupture de ce lien ;
- Créer une liste des voisins qui devraient être informés. Pour chaque nœud de la liste des destinations affectées, prendre les nœuds se trouvant dans la liste des précurseurs, c'est à dire des nœuds qui passent par lui pour atteindre la destination ;
- Envoyer enfin un message erreur de route RERR aux nœuds de la liste précédente. Ce message peut être diffusé (plusieurs précurseurs), envoyé en *unicast* (un seul précurseur) ou envoyé itérativement à plusieurs nœuds en *unicast*.

Chaque nœud recevant le message erreur de route RERR doit marquer les routes indiquées dans le paquet comme invalides tout en recopiant la valeur du numéro de séquence dans sa table de routage pour l'entrée en question. Enfin, le champs durée de vie est changé à la valeur *DELETE_PERIOD*. Donc, une entrée qui n'est plus valide n'est pas effacée immédiatement de la table de routage : elle doit y rester pendant une période ce qui explique la présence de routes invalides dans la table de routage.

Dans cette analyse, nous avons montré que les nœuds se fient directement aux informations présentées par les voisins sans aucune vérification. Ils font confiance aux autres pour se comporter normalement en suivant la spécification du protocole, ce que nous avons formalisé dans cette section par des règles. Ainsi, nous montrons qu'un nœud fait confiance à un autre pour exécuter un ensemble d'actions en des circonstances spécifiques. Mais cette confiance que les nœuds se vouent peut profiter à des nœuds malhonnêtes pour servir leurs intérêts (voir section 2.1).

À partir des règles présentées dans cette section, nous proposons dans ce qui suit de détecter les actions malveillantes : nous introduisons un contrôle du comportement du voisinage (portant sur les actions élémentaires) en vérifiant le contenu de chaque message reçu pour valider l'action du voisin ou lever une alerte. Cette vérification est faite à partir d'un historique de messages en recoupant les informations fraichement reçues avec celles qui sont déjà stockées.

2.3 Détection des nœuds malhonnêtes

Dans cette section, nous nous intéressons à la détection des comportements malveillants en se basant sur les observations des nœuds. Cependant, AODV tel que spécifié ne garde pas assez d'informations pour vérifier les comportements des voisins. Nous proposons ainsi d'enrichir la connaissance de chaque nœud en stockant des informations additionnelles lui permettant de déci-

der de l'honnêteté du voisin. Ce raisonnement repose sur un ensemble de contrôles qui permettent de déceler les incohérences, signes caractéristiques d'actions malhonnêtes, après corrélation des informations reçues avec celles stockées. Nous appelons ces contrôles ***critères d'incohérences***.

Ces critères introduisent de la méfiance envers les nœuds se comportant malhonnêtement. D'après [YKB93], la méfiance introduite concerne une classe d'action ($X \neg trusts_{cc}\ (Y)$). Nous sommes convaincu que cette méfiance doit s'étendre à toutes les classes et ne doit plus être restreinte à une seule classe. Ainsi, si l'action malhonnête d'un nœud est détecté, il ne devrait plus être considéré de confiance pour n'importe quelle action.

Nous représentons les critères d'incohérence comme suit :

Critère d'incohérence 2.x

Identification d'incohérences

$\Rightarrow \quad X \neg trusts\ (Y)$

Les critères obtenus sont composés de deux parties :

- La première vérifie l'existence d'incohérences en recoupant les messages reçus avec ceux stockés ;
- La seconde présente la relation de méfiance résultante ($X \neg trusts\ (Y)$) d'un nœud envers un autre.

Il est à noter que ces critères d'incohérences supposent qu'un nœud malveillant ne peut pas se faire passer pour un autre. Ceci permet d'éviter les fausses accusations : un nœud malveillant peut se comporter malhonnêtement en usurpant l'identité d'un nœud honnête au risque de le faire détecter comme malveillant.

Hypothèse 1. *Un nœud malveillant ne peut pas se faire passer pour un autre.*

Nous avons conscience qu'il s'agit d'une hypothèse importante ayant plusieurs implications. Ceci étant, pour pouvoir la satisfaire, nous proposons d'utiliser un mécanisme d'identité prouvable : il s'agit d'une identité qu'il est facile de vérifier, mais très difficile d'usurper. Par exemple, la clé publique d'une paire de clés publique/privée peut être utilisée comme identité prouvable : un nœud prétendant être identifié par sa clé publique peut signer un *challenge* en utilisant sa clé privée, et est le seul à pouvoir déchiffrer un message qui a été chiffré avec sa clé publique [PA04]. Citons aussi CAM [OR01] et SUCV [MC02] deux mécanismes d'identité prouvable où un nœud se base sur un résumé cryptographique de sa clé publique pour prouver son identité.

Dans ce qui suit, nous présentons tout d'abord les informations additionnelles à stocker pour enrichir la connaissance. Ensuite, nous expliquons comment détecter les comportements malhonnêtes.

2.3.1 Présentation de la table d'historique étendue THE

La table d'historique d'AODV (voir section B.1.2 page 116) a été étendue comme suit :

- Pour les demandes de route (RREQ), les champs identifiant $\#ID$ et adresse de la source $@Src$ sont seulement enregistrés. Nous ajoutons les champs suivants :
  - $\#SNS$: Numéro de séquence de la source qui peut être modifié par un nœud malhonnête qui projette de s'insérer sur la route entre une source et une destination [TBK+03] ;
  - $\#HC$: Nombre de sauts pour atteindre la source ($@Src$) qui, de la même façon que $\#SNS$, peut être modifié par un nœud malhonnête pour s'insérer sur une route ou pour retarder la découverte de route ;
  - ip_src : Adresse IP du nœud qui a envoyé la RREQ et qui peut être différent de l'adresse du nœud ayant initié la demande de route ($@Src$). Cette information servira à traquer les nœuds rejouant des messages.
- Pour les réponses de route (RREP), AODV ne prévoyait pas d'enregistrer les informations les concernant. Nous gardons trace de ce message dans la table d'historique étendue et nous stockons les champs suivants :
  - $@D$: Adresse de la destination de la RREQ dont ce RREP est une réponse ;
  - $@Src$: Adresse de la source de la RREQ dont ce RREP est une réponse ;
  - $\#SND$: Numéro de séquence de la destination qui peut être modifié par un nœud malhonnête pour forcer un nœud honnête à le choisir dans le chemin vers la destination ;
  - $\#HC$: Nombre de sauts pour atteindre la destination ($@D$) qui peut être modifié par un nœud malhonnête qui se veut avoir le chemin le plus court ;
  - ip_src : Représentant l'adresse IP de la source d'un message RREP pris de l'entête IP ;
  - ip_dst : Représentant l'adresse IP de la destination d'un message RREP pris de l'entête IP. Cette adresse combinée à l'adresse de la source ip_src permettent de trouver les nœuds rejouant des messages RREP.

Nous associons à chaque entrée de la table d'historique étendue THE, une durée de vie (*Life Time*) comme c'est le cas pour la table d'historique de base définie dans la spécification d'AODV. Cette durée de vie permet une meilleure gestion de l'espace mémoire pour ne pas saturer un nœud par des informations obsolètes. Notons que pour remplir la table d'historique étendue THE, nous utilisons tous les messages entendus où l'émetteur est à porté et non uniquement les messages où le nœud est la destination.

2.3.2 Critères d'incohérences

Nous présentons les contrôles que les nœuds doivent intégrer afin de vérifier le comportement des voisins pour détecter des nœuds impliqués dans le protocole AODV qui essaient de le contourner pour détourner le trafic en rejouant, fabriquant ou en modifiant des messages.

2.3.2.1 Rejeu de messages

Le rejeu de messages consiste en la retransmission d'anciens messages par un nœud malhonnête. Comme nous l'avons indiqué précédemment, il peut être utilisé pour le détournement du trafic ou la consommation des ressources (batterie, bande passante, etc.). Un ancien message est

un paquet de routage reçu depuis un certain temps et qui n'est plus d'actualité puisqu'il a été traité et que d'autres messages plus récents sont reçus depuis. Dans ce qui suit, nous traitons le rejeu d'une demande de route, d'une réponse de route et d'un message HELLO.

Rejeu de RREQ. En se référant aux règles 2.1 et 2.2, suite à la réception d'une demande de route RREQ et dans le cas où cette demande est déjà traitée, un nœud crée ou met à jour un chemin vers le nœud duquel le message est reçu ensuite efface silencieusement toute demande de route subséquente et liée.

Un nœud malhonnête peut rejouer des demandes de route sans que le voisinage ne s'aperçoit de ce comportement malveillant. Les nœuds cibles (voisinage) continuent à refaire le traitement présenté ci-dessus pour chaque demande de route reçue puisqu'ils font *a priori* confiance à cet émetteur pour se comporter normalement. Cependant, l'action de l'attaquant provoque une consommation de la bande passante par des messages inutiles et des traitements supplémentaires au niveau des nœuds, ce qui dégrade la qualité de services.

L'action de l'attaquant dans ce genre d'attaques repose sur la retransmission d'anciennes demandes de route. Sachant qu'une demande de route est uniquement identifiée grâce à son identifiant ($\#ID$) et l'adresse de la source ($@Src$) et qu'on peut facilement obtenir l'adresse de l'émetteur de chaque demande de route, nous pouvons introduire un contrôle qui vérifie que cette RREQ a été transmise auparavant par ce nœud ou non.

En effet, en stockant dans la table d'historique étendue (THE) l'adresse IP de la source de la RREQ (ip_src) pour chaque demande de route reçue, nous pouvons mettre en place ce contrôle. Il s'agit alors de vérifier à la réception d'une RREQ déjà traitée (la demande de route est dans la table d'historique étendue)

$$M \xleftarrow{\text{RREQ}_N} \ \wedge \ isTreated(RREQ_N, THE_M) = True$$

que l'adresse du nœud émetteur n'est pas la même que celle stockée dans la THE pour toutes les occurrences stockées

$$THE_M.RREQ_N.ip\_src \neq RREQ_N.ip\_src$$

autrement le nœud émettant la RREQ est considéré comme malveillant puisqu'il vient de rejouer un message et ne devrait plus être considéré comme nœud de confiance ($M \neg trusts\,(N)$). Ceci nous amène au premier critère d'incohérence présenté dans l'expression 2.1 :

Critère d'incohérence 2.1

$$\begin{aligned} M \xleftarrow{RREQ_N} \ &\wedge \ isTreated(RREQ_N, THE_M) = True \\ &\wedge \ THE_M.RREQ_N.ip\_src = RREQ_N.ip\_src \\ &\Rightarrow \ M \neg trusts\,(N) \end{aligned}$$

Rejeu de RREP. Comme pour les demandes de route, un nœud malhonnête peut rejouer des réponses de route impliquant la répétition du traitement présenté dans les règles 2.3 et 2.4 au niveau du nœud cible. Toutefois, en utilisant les informations de la table d'historique étendue, à la réception d'une réponse de route ($M \xleftarrow{\text{RREP}_N} N$), un nœud dispose d'une connaissance suffisante pour

vérifier s'il l'a déjà traitée. Il recherche ainsi les réponses de routes similaires reçues précédemment ($isTreated(RREP_N, THE_M) = True$) et vérifie pour chaque occurrence l'adresse IP de la source et de la destination par rapport à celles du paquet en cours de traitement :

$$THE_M.RREP_N.ip\_src = RREP_N.ip\_src \ \wedge \ THE_M.RREP_N.ip\_dst = RREP_N.ip\_dst$$

Si la RREP provient du même nœud et est destinée au même nœud, alors le nœud traitant peut décider de la malhonnêteté de l'émetteur (ip_src) et ne doit plus le considérer comme nœud de confiance ($M \neg trusts\,(N)$). L'expression 2.2 présente ce critère d'incohérence.

Critère d'incohérence 2.2

$$M \xleftarrow{RREP_N} N \ \wedge \ isTreated(RREP_N, THE_M) = True \ \wedge \ THE_M.RREP_N.ip\_src = N$$
$$\wedge \ THE_M.RREP_N.ip\_dst = RREP_N.ip\_dst$$
$$\Rightarrow \ M \neg trusts\,(N)$$

Rejeu de HELLO. Un nœud initialise un message HELLO (envoyé à son voisinage direct) lorsqu'il fait partie d'un chemin actif et s'aperçoit qu'il n'a pas diffusé un message dans le dernier intervalle de temps. Ceci permettra aux nœuds voisins de garder le lien actif.

Étant donné qu'un nœud malhonnête ne peut pas usurper l'identité d'un autre nœud, nous ne voyons pas d'intérêt particulier à ce qu'un attaquant rejoue ses propres messages HELLO.

2.3.2.2 Fabrication de messages

Nous nous intéressons à la détection du comportement d'un nœud malhonnête fabriquant des réponses de route sans avoir reçu et traité la demande de route RREQ correspondante.

La réception d'une réponse de route ($M \xleftarrow{RREP_N} N$) implique d'avoir précédemment traité une demande de route et avoir le chemin inverse vers la source. Donc, si un nœud reçoit une réponse de route qui lui est destinée et ne trouve pas d'entrée vers la source dans sa table de routage,

$$RREP_N.@Src \notin RT_M$$

il déduit que le nœud duquel le message est initié (N) l'a fabriqué. Il ne lui fait plus confiance ($M \neg trusts\,(N)$). L'expression 2.3 présente le critère d'incohérence détectant ce genre de fabrication de messages.

Critère d'incohérence 2.3

$$M \xleftarrow{RREP_N} N \ \wedge \ RREP_N.@Src \notin RT_M$$
$$\Rightarrow \ M \neg trusts\,(N)$$

2.3.2.3 Modification de messages

Un nœud n'est supposé modifier que certains champs en relayant des messages de routage (RREQ et RREP). Nous nous basons sur ce principe pour découvrir les nœuds malhonnêtes ayant changé un champs qui est supposé ne pas l'être.

Modification de RREQ. En relayant une demande de route, le nœud intermédiaire ne peut modifier que le nombre de saut ($RREQ.\#HC$) en l'incrémentant de un et le numéro de séquence de la destination ($RREQ.\#SND$) en gardant la plus grande valeur entre celle reçue dans le paquet et celle trouvée dans la table de routage. Tout changement d'un autre champs peut être considéré comme frauduleux.

Pour pouvoir détecter ce genre de comportement, nous stockons pour chaque demande de route le numéro de séquence de la source ($\#SNS$) et le nombre de saut ($\#HC$) dans la table d'historique étendue.

Ainsi, à la réception d'une demande de route qui a déjà été reçue (d'un autre nœud) et traitée

$$M \xleftarrow{\text{RREQ}_N} \ \wedge\ isTreated(RREQ_N, THE_M) = True$$

mais qui est reçue pour la première fois du voisin N (il s'agit en fait de $RREP_N.ip\_src$)

$$THE_M.RREQ_N.ip\_src \neq N$$

M compare le numéro de séquence de la source reçu dans la RREQ ($RREQ_N.\#SNS$) avec celui trouvé dans sa table de routage ($RT_M.(RREQ_N.@Src).\#SNS$). S'il les trouve différents, il en déduit que cette demande de route a été modifiée.

L'expression 2.4 présente le critère d'incohérence introduisant la méfiance qu'il faut développer à l'envers d'un nœud se comportant de la sorte.

Critère d'incohérence 2.4

$$M \xleftarrow{RREQ_N} \ \wedge\ isTreated(RREQ_N, THE_M) = True \ \wedge\ THE_M.RREQ_N.ip\_src \neq N$$
$$\wedge\ RREQ_N.\#SNS \neq RT_M.(RREQ_N.@Src).\#SNS$$
$$\Rightarrow\ M\ \neg trusts\ (N)$$

Modification de RREP. De même, dans une réponse de route, un nœud n'a le droit de modifier que le nombre de sauts en l'incrémentant de un avant de la retransmettre. Ainsi, nous pouvons déduire que tout autre changement est synonyme de malhonnêteté. En particulier le changement du numéro de séquence de la destination qui donne une indication sur la fraicheur d'une route et la diminution du nombre de saut pour faire croire en un chemin plus court. Ces modifications introduites par un nœud malhonnête provoquent une mise à jour de la table de routage du nœud recevant la RREP par de fausses informations.

L'action malhonnête peut être détectée grâce à l'historique des messages enregistrés. Un nœud

ayant envoyé une réponse de route et qui entend la retransmission du voisin

$$M \xrightarrow{\text{RREP}_M} N \ \wedge \ M \xleftarrow{\text{RREP}_N}$$

peut ainsi contrôler ce que le voisin vient de faire sur le message. Il recherche dans la table d'historique étendue la RREP qu'il a envoyé précédemment et vérifie si le numéro de séquence de la destination n'a pas été altéré (comparer celui reçu $RREP_N.\#SND$ avec celui précédemment envoyé $THE_M.RREP_N.\#SND$). L'expression 2.5 formalise ce cas et incite à se méfier de ces nœuds.

Critère d'incohérence 2.5

$$\begin{gathered}(\ M \xrightarrow{RREP_M} N \ \wedge \ M \xleftarrow{RREP_N} \ ) \ \wedge \ isTreated(RREP_N, THE_M) = True \\ \wedge \ RREP_N.\#SND \neq THE_M.RREP_N.\#SND \\ \Rightarrow \ M \neg trusts\,(N)\end{gathered}$$

L'expression 2.5 ne traite que le cas des nœuds qui sont sur le chemin (relayant la réponse de route) et entendant la retransmission du voisin. Le même raisonnement peut être appliqué sur un nœud qui n'est pas sur la route mais qui est voisin de deux nœuds ou plus sur la route. Il est capable de déduire les modifications du message. Notons X un nœud, voisin de deux nœuds sélectionné dans un chemin M et N. X peut alors entendre la transmission de M et celle de N.

$$M \xrightarrow{\text{RREP}_M} N \ \wedge \ X \xleftarrow{\text{RREP}_M} \ \wedge \ X \xleftarrow{\text{RREP}_N}$$

À la réception de la seconde RREP initiée par N, il peut corréler les informations fraichement reçues avec celles qui existent dans sa table d'historique étendue : la RREP reçue précédemment du nœud M. Ainsi, la comparaison entre les numéros de séquence de la destination ($RREP_N.\#SND$ et $THE_X.RREP_N.\#SND$) permet de détecter toute modification éventuelle de ce champs par le second émetteur (dans ce cas c'est le nœud N).

L'expression 2.6 reprend la précédente (expression 2.5) pour un cas plus général qui s'étend à tout nœud ayant entendu la première transmission et qui voix ensuite la seconde.

Critère d'incohérence 2.6

$$\begin{gathered}(\ M \xrightarrow{RREP_M} N \ \wedge \ X \xleftarrow{RREP_M} \ \wedge \ X \xleftarrow{RREP_N} \ ) \ \wedge \ isTreated(RREP_N, THE_X) = True \\ \wedge \ RREP_N.\#SND \neq THE_X.RREP_N.\#SND \\ \Rightarrow \ X \neg trusts\,(N)\end{gathered}$$

Ainsi, ces critères d'incohérences (2.5 et 2.6) peuvent être aussi utilisés pour détecter la fabrication de réponses de route avec modification du numéro de séquence destination injectées lors du processus de création de route.

En introduisant au niveau de chaque nœud ces contrôles, nous ajoutons une ligne de défense supplémentaire permettant la détection en temps réel de certaines attaques contre les messages de routage et ce à travers la vérification du bon comportement des nœuds par rapport à la spécification du protocole.

2.4 Résumé

Dans ce chapitre, nous nous sommes intéressé à la mise en place d'un mécanisme pour la détection de nœuds malhonnêtes. Le raisonnement adopté est introduit au niveau de chaque nœud de telle sorte qu'il puisse porter un jugement sur le comportement des voisins et décider de sa validité.

Pour y parvenir, nous avons commencé par l'étude des vulnérabilités du protocole AODV afin de caractériser le comportement malveillant. Nous avons présenté ainsi comment les nœuds malhonnêtes opèrent pour aboutir à leur objectif. Cette analyse nous a permis de montrer que les actions malveillantes se basent essentiellement sur la combinaison d'actions élémentaires (rejeu, modification, suppression, fabrication) portant sur les messages de routage en exploitant la confiance que les nœuds se vouent. Ceci nous a amené à analyser le comportement normal du protocole pour trouver les moyens qui permettrons de détecter ces actions.

Nous avons utilisé le langage présenté par [YKB93] pour la description du comportement normal ce qui nous a permis d'expliciter les relations de confiance implicite qui existent entre les nœuds. Cette analyse a montré que les nœuds se fient directement aux informations présentées par les voisins sans aucune vérification : ils font confiance aux autres pour se comporter normalement en suivant la spécification du protocole. Cependant, cette confiance que les nœuds se vouent peut profiter à des nœuds malhonnêtes pour servir leurs intérêts.

Les règles de confiance associées à l'analyse du comportement malveillant nous a permis de déceler des incohérences signes caractéristiques d'actions malhonnêtes. Cependant, la connaissance acquise par un nœud ne permet pas de vérifier le comportement et ainsi de déceler les incohérences. Ainsi, nous avons enrichi cette connaissance et nous avons mis en place un ensemble de contrôles qui permettent de déceler les incohérences dans les annonces après corrélation des informations reçues avec celles stockées.

Dans le chapitre suivant, nous implémentons ce raisonnement au niveau de chaque nœud. Et pour pouvoir évaluer les performances du système de détection, il faut pouvoir le mettre à l'épreuve ce qui nécessite la mise en place de comportements malhonnêtes. L'évaluation portera ensuite sur les performances du système de détection ainsi que sur celle du protocole pour montrer son efficacité de détection sans pour autant dégrader les performances du protocole de base.

CHAPITRE 3

EXPÉRIMENTATIONS DU RAISONNEMENT BASÉ SUR LA CONFIANCE

Nous avons présenté précédemment les concepts théoriques relatifs à la détection d'actions malhonnêtes se basant sur les critères d'incohérences. Cette détection passe par une étude détaillée du comportement de chaque voisin qui permet pour chaque nœud de déceler toute déviation par rapport au comportement normal décrit dans la spécification d'AODV.

Dans ce chapitre, nous nous intéressons à l'évaluation de notre proposition par l'intermédiaire de mesures de performances sur différents scénarios de mobilité, de mouvement et de trafic. Dans ce qui suit, nous présentons le choix de l'environnement de simulation (section 3.1). Nous argumentons ensuite (section 3.2) nos choix sur l'implémentation AODV. Puis, nous détaillons l'implémentation de la solution.

Pour tester la résistance de notre modèle de détection, nous mettons en place des exemples d'attaques dans la section 3.4. Nous présentons notamment les détails techniques de leur mise en place par un nœud malhonnête.

Enfin, après avoir décrit dans la section 3.5 les différents paramètres à prendre en considération pour mettre en place les simulations, nous montrons l'efficacité de la solution à travers les résultats obtenus des simulations (section 3.6). Nous évaluons la performance du système de détection montrant sont efficacité à contrer les actions malhonnêtes. Ensuite, nous évaluons la performance du protocole pour montrer que notre mécanisme ne dégrade pas la qualité de service. Enfin, nous testons les limites de notre mécanisme en observant le comportement des nœuds face à l'augmentation du nombre d'attaquants dans le réseau.

3.1 Environnement de simulation

Au cours des années 90, plusieurs simulateurs ont vu le jour tels que NS-2 [IH08] (*Network Simulator version 2*), *GloMoSim* [BTA+99] et *OPNET* [Cha99]. NS-2 reste le simulateur de réseau le plus utilisé dans le milieu académique ainsi que dans l'industrie.

NS-2 est un outil gratuit de simulation par événements discrets [1] dont le code source est disponible. Il fournit les mécanismes nécessaires à la mise en œvre des protocoles et la simulation de leurs comportements. NS-2 a gagné en popularité dans la communauté de recherche en réseau depuis sa naissance en 1989 grâce à sa flexibilité et son caractère modulaire. Plusieurs révisions ont vu le jour marquant la maturité croissante de l'outil tout en maintenant sa solidité et sa polyvalence. Il combine le langage de script OTcl et le langage C++. L'interpréteur OTcl sert à exécuter les scripts de commande utilisateur qui permettent la configuration, la description et la mise en place des simulations alors que C++ est utilisé pour l'implémentation du noyau du simulateur ainsi que des protocoles.

Une simulation suit le processus suivant :

Phase 1. Configuration du réseau : il s'agit de créer et de configurer les composants du réseau (nœuds, connexions entre les nœuds, type de connexion, durée d'échange, le début du transfert de données, ...). Pour ce faire, l'utilisateur créé un script OTcl (`<nom_script>.tcl`) contenant les commandes nécessaires pour la mise en place de la simulation.

Phase 2. Exécution de la simulation : il s'agit d'exécuter la simulation configurée dans la phase 1. Une horloge de simulation est maintenue pour exécuter les événements chronologiquement jusqu'à atteindre une limite spécifiée précédemment dans la phase de configuration. Pour cela, NS-2 offre aux utilisateurs une commande exécutable (`ns`) qui prend le nom du fichier contenant le script de simulation (`<nom_script>.tcl`) comme argument en entrée et produit en sortie deux fichiers de traces :

- `<nom_script>.tr` dans lequel sont inscrites toutes les actions observées lors de la simulation ;
- `<nom_script>.nam` qui sert à la création de l'animation correspondante à la simulation grâce au programme NAM (*Network AniMator*) inclu dans NS-2 et ce en exécutant la commande `nam` suivie du fichier à visualiser.

Phase 3. Évaluation de la performance du réseau simulé : il s'agit de collecter et de compiler les résultats des simulations. Un parcours du fichier de traces (`<nom_script>.tr`) s'impose pour dégager les informations voulues. Des langages de script (tel que `awk` ou `shell`) sont généralement utilisés pour parser les fichiers et dégager l'information voulue.

La figure 3.1 met en avant de manière schématique les composants les plus importants de NS-2

1. Dans les simulations par événements, les opérations d'un système sont représentées sous forme d'une séquence chronologique d'événements. Chaque événement se produit à un instant donnée et marque le changement d'état du système. En d'autres termes, le simulateur va passer d'un événement au suivant en mettant à jour à chaque fois la description de l'état du système. Et puisqu'il ne se passe rien entre les événements (pas d'évolution significative des variables), la date saute de la date courante à celle du nouvel événement. Si un tel système admet une description selon des variables évoluant de manière discontinue alors il est dit système discret.

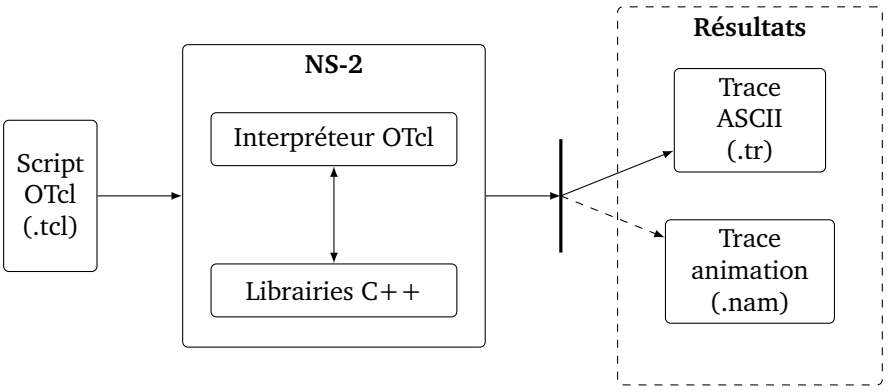


FIGURE 3.1 – Représentation schématique de NS-2

ainsi que l'entrée à fournir à la commande `ns` et les sorties obtenues après exécution. Il est à noter que les sorties présentées sont celles obtenues par défaut. Nous pouvons cependant paramétrer le script `.tcl` (donnée en entrée) en précisant l'information qui devrait être inscrite dans les fichiers de sortie ou aussi les remplacer par d'autres fichiers de trace (dont le contenu et la forme sont spécifiés par l'utilisateur).

Pour conclure, NS-2 offre un compromis entre performance et facilité d'utilisation. Ainsi, nous avons adopté cet outil pour l'implémentation et la simulation de notre proposition.

3.2 Choix de l'implémentation d'AODV

L'implémentation d'AODV inclue par défaut dans NS-2 n'est pas conforme à la RFC 3561. Ainsi, nous avons choisi d'utiliser l'implémentation AODV-UU [Nor07] dont les principales caractéristiques sont présentées ci-après :

- Conformité à la RFC 3561 ;
- Fonctionne avec les noyaux linux 2.4 et 2.6 ainsi que dans le simulateur NS-2 ;
- Installation possible sur des appareils à base de processeurs ARM/MIPS (par exemple iPAQ, Zaurus, N900, iPhone) ainsi que sur quelques routeurs comme le LinkSys WRT54G ;
- Détection et évitement des liens unidirectionnels en option.

Pour comprendre le fonctionnement de cette implémentation, nous présentons le traitement fait par AODV-UU à la réception d'un paquet. Il est particulièrement intéressant de noter le cheminement d'un paquet entre les fonctions (imbrication des appels) afin de comprendre le fonctionnement et ainsi de trouver sur quelle méthode intervenir pour ajouter les contrôles qui permettront de détecter les comportements malhonnêtes. La figure 3.2 montre les appels de méthodes lors de la réception d'un paquet.

Lors de la réception d'un paquet au niveau de la couche liaison et après vérification qu'il s'agit d'un paquet AODV-UU, il y a invocation de la méthode du protocole permettant le traitement du paquet au niveau de la couche réseau : il s'agit de la méthode `recv()` de la classe `aodv-uu.cc` qui se charge de vérifier la validité du paquet. Elle invoque `processPacket()` ou `recvAODVUUPacket()`

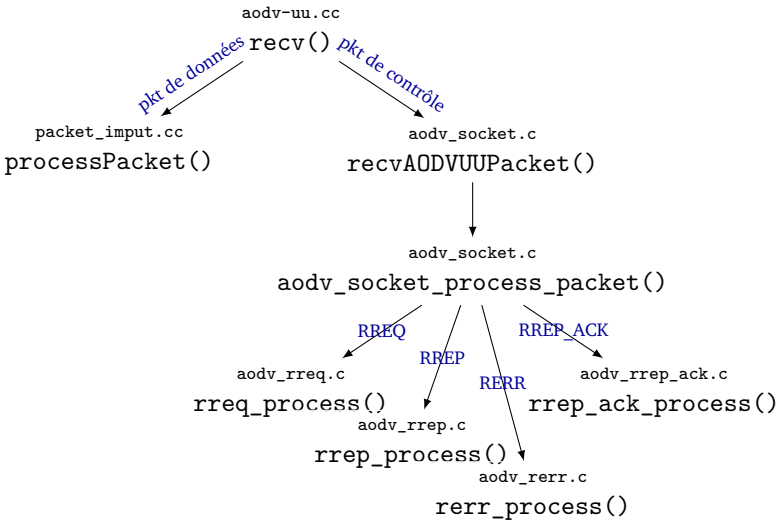


FIGURE 3.2 – Imbrication des appels de méthodes dans AODV-UU

selon qu'il s'agit d'un paquet de données ou d'un paquet de contrôle. Si c'est un paquet de contrôle, `recvAODVUUPacket()` fait appel à la fonction `aodv_socket_process_packet()` pour aiguiller le paquet selon le type du message de contrôle vers la méthode qui présente le traitement adéquat :

- `rreq_process()` s'il s'agit d'une demande de route ;
- `rrep_process()` s'il s'agit d'une réponse de route ;
- `rerr_process()` s'il s'agit d'une erreur de route ;
- `rrep_ack_process()` s'il s'agit d'un accusé de réponse de route.

Ces quatre fonctions présentent la totalité du traitement des messages de contrôle reçus. Une étude plus approfondie du code nous a permis de repérer les emplacements adéquats où nous ajoutons notre système de détection. C'est pourquoi nous nous intéressons à l'implémentation de ces fonctions. L'annexe C présente à titre d'exemple le pseudo-code des fonctions `rreq_process()` et `rrep_process()`.

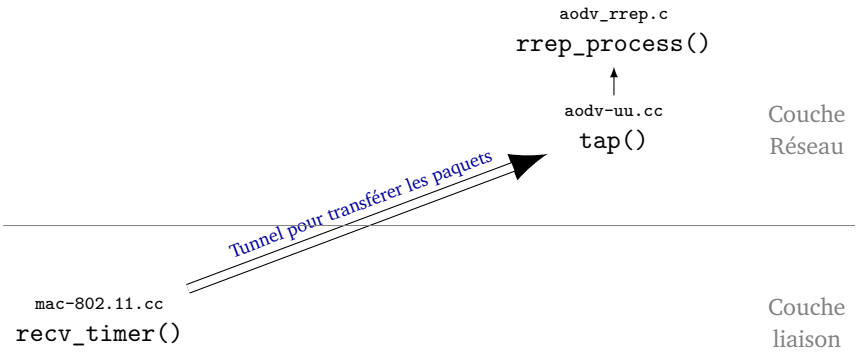


FIGURE 3.3 – Ajout de la fonction `tap()`

Comme pour les solutions à base de *watchdog*, nous devons activer le mode *promiscuous* pour pouvoir remonter les messages de réponse de route RREP (qui rappelons le sont émis en *unicast*) de la couche MAC vers la couche réseau. Dans NS-2, cette activation passe par l'ajout de la fonction `tap()` au niveau de la classe `aodv-uu` et la configuration de la classe 802.11 pour remonter les paquets reçus à cette fonction (voir figure 3.3).

L'activation de la fonction `tap()` permet d'acheminer tout paquet de contrôle AODV-UU depuis la couche liaison (MAC) vers la couche réseau. Les paquets sont alors remontés en utilisant la procédure normale mais aussi à travers la fonction `tap()`. Ainsi, un filtrage de paquet s'impose pour ne pas traiter un paquet plus d'une fois : nous devons laisser passer les paquets de RREP que le nœud peut entendre et qui ne lui sont pas destinés (nœud traitant) ; tout autre paquet est à supprimer puisqu'il s'agit d'un doublon. Les paquets acceptés sont redirigés vers `rrep_process`.

3.3 Implémentation des critères d'incohérences

Dans cette section, nous nous intéressons à la mise en place de notre système de protection contre les nœuds malveillants. Ce système se base essentiellement sur des contrôles de validité du comportement réalisé juste avant de faire le traitement normal du protocole de base.

Nous avons mis en place un mécanisme permettant de simuler dans le même réseau des nœuds exécutant AODV de base (sans modifications) et des nœuds exécutant la version modifiée que nous proposons et ce dans le but de voir leurs comportements respectifs. Ainsi, nous ajoutons une variable entière d'initialisation du comportement du nœud et qui est utilisée dans le script d'initialisation de la simulation lors de la création des nœuds. Nous dénotons cette variable `reasoning_mode` indiquant si le mode de raisonnement est activé (valeur de la variable égale à un) ou non (valeur de la variable égale à zéro).

Donc, tout ce que nous allons présenter dans cette section concernant le raisonnement que nous introduisons doit être placé obligatoirement dans le contrôle suivant :

```
// Le code ajouté doit être à l'intérieur de ce contrôle
Si (reasoning_mode == 1) Alors
    // Traitement que nous ajoutons concernant le contrôle de
    // validité du comportement qui correspond à
    // l'implémentation des critères d'incohérences
```

Rappelons que nous essayons de détecter les incohérences en recoupant les informations fraichement reçues dans les messages de contrôle avec celles stockées dans l'historique. Les informations actuellement stockées dans la table d'historique sont insuffisantes pour pouvoir décider de la malhonnêteté (ou la culpabilité) d'un nœud, c'est pourquoi il nous a fallu étendre ces informations et stocker tous les messages entendus dans la Table d'Historique Étendue (THE).

- Pour les demandes de route RREQ :
  - Ajouter les données manquantes dans la table d'historique. Le code correspondant s'insère au niveau de la ligne 10 de l'algorithme C.1 (p.122) ;
  - Stocker les messages qui sont normalement ignorés. Il s'agit des RREQ qui sont déjà traitées

et qui sont reçues une autre fois d'un autre nœud. Cet ajout doit s'opérer au niveau du bloc d'instructions numéro 7 de l'algorithme C.1 (p.122).

– Pour les réponses de route RREP : Stocker toutes les RREP reçues. Le code correspondant à cet ajout doit s'opérer au niveau du bloc d'instructions numéro 7 de l'algorithme C.2 (p.124).

De cette manière, nous avons la possibilité de vérifier certaines informations en recoupant les données reçues avec celles stockées. Cette vérification est faite par l'intermédiaire d'un ensemble de règles permettant la détection des comportements malhonnêtes que nous avons présenté dans le chapitre 2. Dans ce qui suit, nous reprenons ces critères pour expliquer comment les mettre en place dans le code. Il est à noter que ces critères sont implémentés sous forme de contrôles et sont insérés à l'intérieur de la structure de contrôle qui commence à la ligne 6 : il s'agit du bloc d'instructions 7 de l'algorithme C.1 (p.122) pour les RREQ et juste après la structure de contrôle qui commence à la ligne 5 (bloc d'instructions numéro 7) de l'algorithme C.2 (p.124) pour les RREP.

Critère d'incohérence 3.3.1

« Un nœud n'est pas supposé re-diffuser une RREQ qu'il a déjà diffusée » : un nœud peut facilement vérifier ce critère en consultant sa THE à la réception d'une RREQ pour voir s'il a déjà reçu la même demande de route provenant du même émetteur auquel cas il est en mesure de déclarer un rejeu de RREQ perpétré par le nœud émettant le message.

```
// Le nœud M reçoit une RREQ provenant du nœud N
Si ((RREQ ∈ THE) && (ip_src de la RREQ trouvé dans la THE == N)) Alors
    N a rejoué une RREQ
```

Critère d'incohérence 3.3.2

« Un nœud n'est pas supposé renvoyer en *unicast* un message RREP qu'il a déjà envoyé » : la vérification de ce critère peut être faite à la réception d'une réponse de route en consultant sa THE à la recherche d'une RREP équivalente provenant du même nœud.

```
// Le nœud M reçoit une RREP du nœud N
Si ((RREP ∈ THE) && (ip_src de la RREP trouvé dans la THE == N)) Alors
    N a rejoué une RREP
```

Critère d'incohérence 3.3.3

« Un nœud n'est pas supposé envoyer une RREP s'il n'a pas reçu la RREQ correspondante » : si ce critère n'est pas vérifié c'est que l'émetteur du message est en train de fabriquer des RREP. Ainsi, à la réception d'une RREP, le nœud vérifie l'existence de la RREQ correspondante dans sa THE et vérifie aussi qu'il n'existe aucune RREP concernant la destination (@D) qui a été traitée précédemment par le nœud émetteur. Ceci prouve que le nœud initialisant la RREP n'a pas d'entrée pour la destination dans sa table de routage et que, de plus, la RREQ correspondante n'a pas été traité auparavant.

```
// Le nœud M reçoit une RREP du nœud N
Si ((Pas de RREQ dans la THE correspondant au RREP reçue)
        && (∄ RREP concernant @D dans la THE avec ip_src==N)) Alors
    N a fabriqué une RREP
```

Critère d'incohérence 3.3.4

« Un nœud n'est autorisé à modifier que le champs nombre de sauts (`#HC`) et le champs numéro de séquence destination (`#SND`) en relayant une RREQ » : tout nœud peut superviser l'action des voisins sur un message qu'il a précédemment traité ou vu passer et détecter tout changement non autorisé.

```
// Le nœud M reçoit/entend une RREQ du nœud N
Si ((RREQ ∈ THE) && (#SNS reçu ≠ #SNS trouvé dans la THE)) Alors
    N a modifié le numéro de séquence de la source dans la RREQ relayée
```

Critère d'incohérence 3.3.5

« Un nœud ne modifie que le nombre de sauts `#HC` en relayant une `RREP` » : ce critère permet de vérifier si la réponse de route entendue n'a pas subi des modifications non autorisées lors de sa retransmission.

```
// Le nœud M reçoit/entend une RREP du nœud N
Si ((RREP ∈ THE) && (#SND reçue ≠ #SND trouvé dans la THE)) Alors
    N a modifié le numéro de séquence de la destination dans la RREP relayée
```

La mise en place de ces contrôles représente une ligne de défense supplémentaire qu'un nœud peut mettre en place contre les actions malhonnêtes de certains nœuds sans pour autant perdre son efficacité comme nous allons le démontrer dans la section 3.6. Avant cela, nous présentons dans la section suivante l'implémentation des attaques que nous avons mis en place pour tester notre système de détection.

3.4 Implémentation du comportement malhonnête

Comme nous l'avons expliqué dans le chapitre précédent (section 2.1), un attaquant peut utiliser les vulnérabilités du protocole pour aboutir à ses fins. Il se base essentiellement sur des actions élémentaires (qu'il peut éventuellement composer) et abuse ainsi de la confiance qui lui a été accordée par son voisinage. Nous montrons à travers cette section comment les attaques sont modélisées.

Pour simplifier la mise en place d'attaques au niveau des simulations et pour les rendre configurables à travers les scripts d'initialisation, nous avons introduit une seconde variable entière dénotée `node_behavior` et qui sera utilisée dans le script de simulation lors de la création des nœuds.

Cette variable permet de paramétrer le comportement du nœud en spécifiant s'il s'agit d'un nœud malhonnête (valeur différente de zéro) ou non (égale à zéro). Ainsi, chacun des comportements malhonnêtes suivants aura une valeur différente.

Attaque 3.4.1 (*Rejeu de demande de route*)

À chaque fois que le nœud malhonnête retransmet une demande de route RREQ, il choisit au hasard une autre RREQ précédemment traitée et la rejoue.

Cette attaque s'insère dans deux endroits différents : après les instructions de la ligne 17 et 23 de l'algorithme C.1 (p. 122). Cette action se déroule de la sorte :

Si *(node_behavior == val1)* **Alors**
    Choisir une requête à répliquer au hasard
    Si *(La requête choisie est différente de la RREQ actuellement traité)* **Alors**
        Ré-envoyer cette requête

Attaque 3.4.2 (*Rejeu de réponse de route*)

Comme pour les demandes de route, le nœud malhonnête choisit au hasard une réponse de route RREP diffusée auparavant qu'il retransmet en plus de la RREP qu'il vient de recevoir et de traiter.

Cette attaque s'insère au niveau des instructions de la ligne 18 de l'algorithme C.2 (p. 124). Cette action se déroule comme suit :

Si *(node_behavior == val2)* **Alors**
    Choisir une réponse de route à répliquer au hasard
    Reconstituer le RREP à retransmettre
    Déterminer le nœud vers lequel la RREP doit être acheminée
    Ré-envoyer cette RREP

Attaque 3.4.3 (*Modification du numéro de séquence de la source #SNS dans une RREQ*)

L'attaquant modifie le numéro de séquence de la source #SNS dans la demande de route RREQ reçue avant de la retransmettre. Ce code s'insère juste avant les instructions des lignes 17 et 23 de l'algorithme C.1 (p. 122). Voici le pseudo-code correspondant à cette modification :

Si *(node_behavior == val3)* **Alors**
    Augmenter ou diminuer le numéro de séquence de la source #SNS

Attaque 3.4.4 (*Modification du numéro de séquence de la destination #SND dans une RREP*)

Comme dans l'attaque 3.4.3, le nœud malhonnête modifie le numéro de séquence de la destination #SND dans la réponse de route RREP reçue avant de la retransmettre. Il intervient sur un champs qu'il n'a pas le droit de modifier. Les lignes suivantes s'insèrent juste avant les instructions de la ligne 18 de l'algorithme C.2 (p. 124).

```
Si (node_behavior == val4) Alors
    Augmenter ou diminuer le numéro de séquence de la destination #SND
```

Ces quatre premières attaques se basent sur des actions élémentaires dans le but de perturber le trafic, augmenter le traitement au niveau des nœuds ou retarder l'établissement d'un chemin. Ces attaques sont des attaques très simples que nous avons avant tout implanté pour vérifier le fonctionnement de base de notre système de détection.

Les attaques que nous présentons dans la suite combinent plusieurs actions élémentaires et diffèrent des précédentes puisqu'elles ont un but extrêmement précis : l'attaquant souhaite s'insérer sur le chemin entre la source et la destination. Notons que dans ces attaques, les actions malhonnêtes portent exclusivement sur les réponses de route RREP.

Attaque 3.4.5 (*Fabrication de RREP avec un #SND grand dès la réception d'une RREQ*)

Dès la réception d'une RREQ, le nœud malhonnête répond par une réponse de route avec un #SND grand pour garantir que sa réponse soit prise en compte. Cette action est faite même s'il n'a pas de chemin valide vers la destination demandée. Le pseudo-code suivant est à insérer au niveau des instructions de la ligne 17 et 23 de l'algorithme C.1 (p. 122).

```
Si (node_behavior == val5) Alors
    Si (∃ chemin valide vers la destination) Alors
        Fabriquer une RREP avec #SND > à celui existant dans la RT
    Sinon
        Fabriquer une RREP avec #SND grand pris au hasard
```

Attaque 3.4.6 (*Fabrication de RREP avec un #SND grand en entendant une RREP*)

Dès la réception d'une réponse de route qui ne lui est pas destiné mais qu'il a entendue, le nœud malhonnête fabrique une réponse de route avec un #SND plus grand que celui qu'il a entendu pour garantir que sa réponse soit prise en compte. Le pseudo-code suivant est à insérer avant l'instruction de la ligne 9 de l'algorithme C.2 (p. 124).

```
Si (node_behavior == val6) Alors
    // En entendant cette RREP, je suis voisin de ip_src
    Si (∃ chemin valide vers ip_dst) Alors
        Fabriquer une RREP avec #SND plus grand et l'envoyer à ip_dst
    Sinon
        Fabriquer une RREP avec #SND plus grand et l'envoyer à ip_src
```

De cette manière, il suffit d'initialiser la variable `node_behavior` lors de la création du nœud selon le type d'attaque à effectuer (`val1 ...val6`) dans le script (`.tcl`) pour simuler un nœud malhonnête. Un nœud honnête met à zéro cette variable. Ainsi, nous pouvons simuler très simplement dans le même réseau des nœuds honnêtes et malhonnêtes.

3.5 Mise en place des simulations dans NS-2

Nous présentons dans cette section les différents paramètres à prendre en considération dans les simulations. Ces paramètres sont initialisés dans le script (<nom_script.tcl>). Nous décrivons ainsi le contenu de ce fichier regroupant les paramètres des simulations qui concernent la topologie, le trafic, la mobilité, le *timing* des événements, etc.

Topologie. Nous considérons un réseau ad hoc composé d'un nombre de nœuds mobiles variant de 20 à 200 nœuds placés au hasard sur un terrain dégagé de 1000 m $\times$ 600 m. Les nœuds utilisent le protocole MAC IEEE 802.11 et la bande passante est de 2 Mbps.

Structure des nœuds. Chaque nœud maintient une file d'attente de type FIFO (premier entré, premier sorti). Ainsi, les messages les plus anciens sont effacés lors d'un débordement dans cette file. La taille de cette file d'attente est fixée à 50 paquets.

Mobilité. Les nœuds se déplacent constamment en utilisant le modèle de cheminement *random waypoint* : un scénario de mouvement est généré aléatoirement et les nœuds se déplacent linéairement avec une vitesse constante comprise entre 0.0 et 20.0 m/s de sa position jusqu'à la destination choisie. Une fois arrivé, il met le cap vers une nouvelle destination sans marquer un temps de pause et ainsi de suite jusqu'à la fin du temps de simulation. Nous utilisons le script `setdest`, fourni avec NS-2, qui permet la génération automatique de ce type de scénario.

Modèle du trafic. Chaque simulation dure 500 secondes durant laquelle un certain nombre de pairs de nœuds veulent échanger des paquets de données de type CBR (Constant Bit Rate). La taille des paquets échangés est de 512 octets. Ces paquets sont diffusés à intervalles réguliers (tous les 0.25 s) entre le début et la fin de la transmission. Les scénarios de transmission sont générés automatiquement en utilisant le script `cbrgen.tcl` fourni avec NS-2.

Le tableau 3.1 présente les différents paramètres utilisés dans les simulations.

Modèle de propagation sans fil	Free Space
Type d'antenne	Omni-directional
Modèle de mobilité	Random waypoint
Protocole MAC	802.11
Gestion et ordonnancement de la file d'attente	DropTail/PriQueue
Capacité de la file d'attente	50 packets
Placement des nœuds	Random
Nombre de nœuds	20, 40, 60, 80, 100, 200
Nombre de nœuds malhonnêtes	1, 5, 10, ...
Vitesse Minimale/Maximale	0 - 20 m/s
Temps de pause	0 s
Dimension du terrain	1000 m x 600 m
Nombre de paires (source, destination)	7, 15, 20, 30, 40 ... 200
Taille des paquets de données	512 bytes
Intervalle pour l'envoi des paquets de données	4 packet/second

TABLE 3.1 – Paramètres de simulation dans NS-2

Une fois que ces scripts de simulations ainsi que les fichiers contenant les scénarios de mobilité et de trafic sont prêts, il suffit de créer un script `shell` qui permet d'automatiser le lancement des simulations. Une fois l'exécution des simulations terminée, nous analysons les fichiers de trace (en utilisant des scripts `awk`) afin de filtrer les informations pertinentes et nous synthétisons les résultats obtenues sous forme de courbes. Dans la section suivante, nous présentons ces courbes et nous interprétons leur signification.

3.6 Résultats des simulations

Cette section est divisée en trois parties distinctes. Dans la première, nous nous intéressons à la performance du système de détection en terme de taux de détection d'actions malhonnêtes et de taux de faux-positifs. Dans la deuxième partie, nous nous intéressons aux performances du protocole. Nous montrons ainsi que l'ajout de notre mécanisme n'a pas d'impact négatif significatif sur les performances du protocole de base. Enfin dans la troisième partie, nous évaluons la performance du système de détection face à l'augmentation du nombre de nœuds malhonnêtes.

3.6.1 Performance du système de détection

Pour montrer l'efficacité de notre système de détection, nous définissons le ratio de détection correspondant au nombre de voisins d'un nœud malhonnête détectant un comportement frauduleux sur le nombre total des voisins de ce nœud. Cette métrique montre combien de nœuds dans le voisinage d'un nœud malhonnête peuvent découvrir l'action malhonnête.

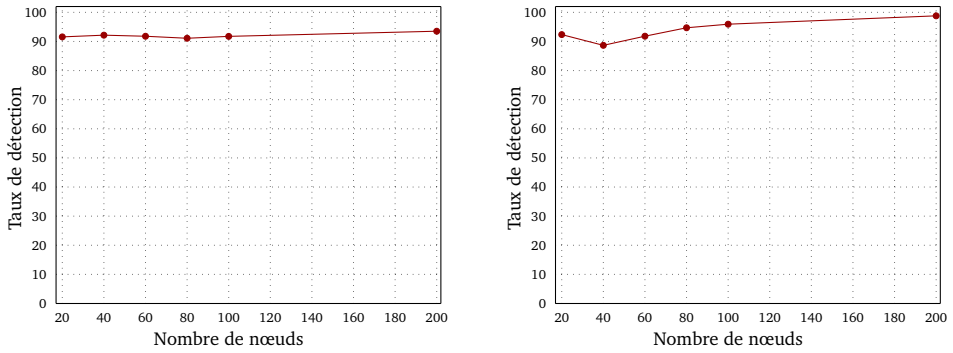
$$Taux\,de\,détection\,(N_i) = \frac{voisins\,de\,N_i\,détectant\,attaque}{total\,des\,voisins\,de\,N_i} \times 100$$

D'autres formules existent pour mesurer le taux de détection telle que « nombre d'attaques détectées par au moins un nœud divisé par le nombre total d'attaques ». Les résultats obtenus par ces formules sont plus généreux en terme de pourcentage de détection que celle que nous avons adopté. Nous justifions ce choix par le fait que notre mécanisme se base essentiellement sur le voisinage direct pour détecter tout comportement malhonnête, ce qui permet de stopper la propagation de l'attaque. Ainsi, le ratio choisi nous semble plus pertinent par rapport aux autres.

Nous présentons dans un premier temps l'analyse liée à la métrique choisie pour les attaques élémentaires. Ensuite, nous présentons celle correspondant aux attaques complexes. Nous terminons par une discussion concernant les fausses alertes (faux-positifs).

3.6.1.1 Résultats concernant les attaques élémentaires.

La figure 3.4 montre l'évolution du taux de détection en variant le nombre de nœuds de 20 à 200. L'intensité du trafic et la mobilité restent les mêmes pour toutes les tailles de réseaux. Il est à noter que la variation du nombre de nœuds correspond à la variation de la densité du réseau : par exemple un réseau de 60 nœuds correspond à une densité de 100 nœuds par kilomètre carré (nœud/km^2).



a. Rejeu de RREQ

b. Modification du `#SNS` dans une RREQ

c. Rejeu de RREP

d. Modification du `#SND` dans une RREP

FIGURE 3.4 – Impact de la variation de la densité du réseau (trafic et nombre d'attaquants constants)

Les résultats montrent un taux de détection supérieur à 90% pour les attaques contre les demandes de route RREQ (figures 3.4a et 3.4b). Après analyse, nous avons établi que les cas de non-détection sont dûs essentiellement à la mobilité des nœuds. Notamment, les cas de non-détection sont dûs au partitionnement du réseau. En effet, les nœuds du réseau sont constamment en mouvement et le système de détection se base sur les observations collectées à propos des nœuds voisins : lorsqu'un nœud malhonnête arrive pour la première fois dans le voisinage, il peut ne pas être détecté.

Les figures 3.4c et 3.4d présentent l'évolution du taux de détection pour les attaques sur les réponses de route RREP. Pour ces attaques, le taux de détection varie entre 70 et 95%. Si les cas de non-détection sont encore une fois dûs à la mobilité, la baisse du taux de détection par rapport aux attaques contre les RREQ est ici essentiellement dû à la non-diffusion des RREP dans le réseau qui, rappelons le, sont envoyées en *unicast*. Ainsi, seuls les nœuds ayant envoyé précédemment la réponse de route ou ceux ayant écouté la transmission du voisin sauront détecter un comportement malhonnête. Parallèlement, la baisse constatée sur la figure 3.4c tient aux faits que :

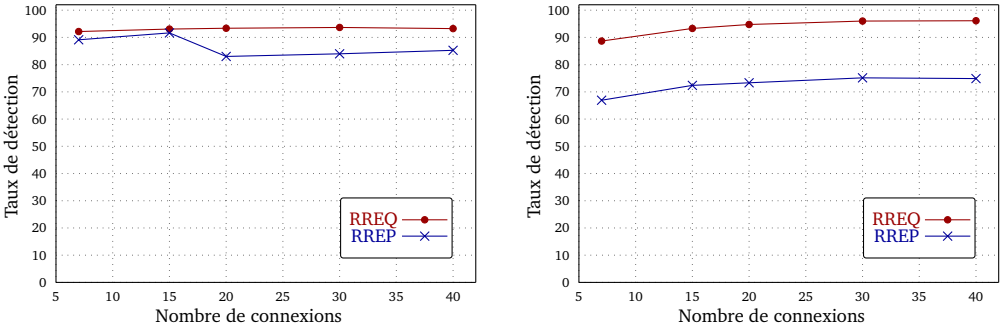
– Lorsqu'il y a moins de nœuds, l'attaquant est sur plus de routes et rejoue donc plus de messages RREP. En effet, d'une part la diminution de la densité du réseau fait que les possibilités d'établissement d'un chemin sont plus rares ce qui fait que si l'attaquant est bien placé, il peut participer dans plus de routes (statistiquement). D'autre part, l'implémentation de l'attaque

est de telle sorte qu'un nœud malveillant ne rejoue des RREP que s'il est proche d'un chemin, ce qui lui donne la possibilité de rejouer plus de messages et augmente par la même occasion le risque de détection de son action malhonnête.

– Plus la densité est grande, plus l'attaquant a de voisins, qui du fait du changement brusque de voisinage liée à la mobilité, n'ont pas entendu la première émission de la RREP et qui ne détecteront pas l'attaque.

Nous avons testé l'impact de la variation de la densité du trafic définie par la variation du nombre de connexions sur le taux de détection des attaques élémentaires dans deux réseaux de densités fixes :

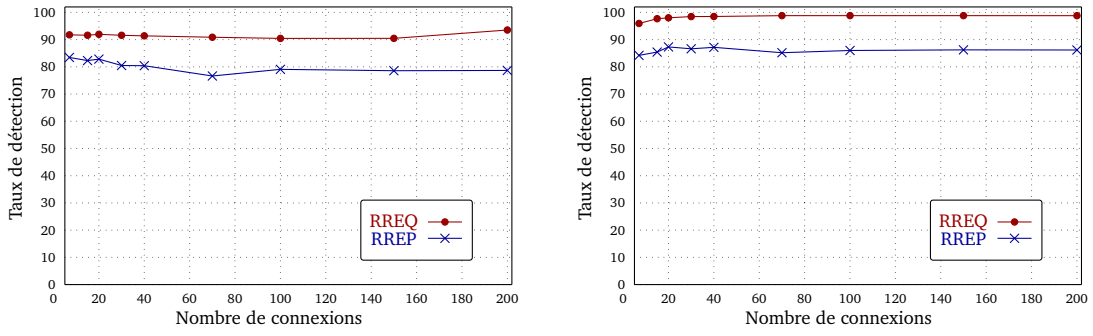
– Le premier est de 40 nœuds (66 nœud/km^2) dans lequel nous avons fait varier le nombre de connexions (communication entre une source et une destination) de 7 à 40. Par exemple, pour 40 connexions, chaque nœud établie une connexion avec un autre nœud pris au hasard dans le réseau ;



a. Rejeu b. Modification

FIGURE 3.5 – Impact de la variation du trafic dans un réseau de 40 nœuds

– La taille du second réseau est 100 nœuds (166 nœud/km^2) dans lequel on fait varier le nombre de connexions de 7 à 200 (pour 200 connexions, chaque nœud établie en moyenne deux connexions avec deux nœuds pris au hasard dans le réseau).



a. Rejeu b. Modification

FIGURE 3.6 – Impact de la variation du trafic dans un réseau de 100 nœuds

Les figures 3.5 et 3.6 confirment les résultats obtenus en variant la densité du réseau : i) plus la densité du trafic est importante, plus le taux de détection devient grand, ii) le taux de détection des attaques concernant les demandes de route RREQ est plus grand que le taux de détection des attaques concernant les réponses de route RREP et, iii) la non-détection est dû en premier lieu au changement de voisinage ainsi que la non-dissémination des réponses de routes par rapport aux demandes de route.

Nous remarquons par contre certaines particularités relatives à ces simulations :

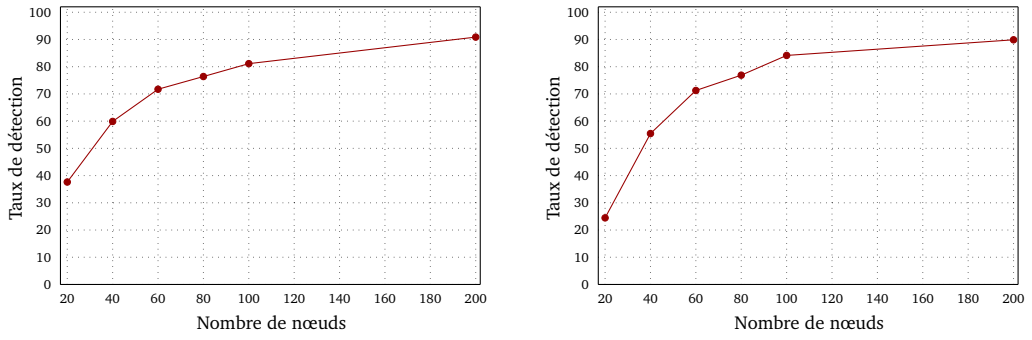
– Après une légère augmentation au début, les courbes de la figure 3.5 et 3.6 semblent se stabiliser autour d'une valeur (différente pour chaque attaque/taille de réseau) malgré une densité de trafic croissante.

– Le taux de détection dans un réseau de 100 nœuds (figure 3.6) est supérieur au taux correspondant dans un réseau de 40 nœuds (figure 3.5) et ce pour la même densité de trafic. Ceci s'explique par le fait que plus le réseau est dense plus le nœud malhonnête à de voisins qui risquent de détecter son comportement frauduleux.

Malgré le fait que nos règles de détection semblaient totalement correctes de manières intuitives, ces tests nous ont permis de constater que dans des cas réels de mobilité, un attaquant peut ne pas être détecté par tous ses voisins. Néanmoins ces cas sont rares et le taux de détection reste élevé : il dépasse les 90% dans la plupart des cas. Nous remarquons que ce taux augmente avec la variation de la densité du trafic et celle du réseau. Nous avons constaté aussi que le taux de détection des attaques contre les RREQ est légèrement supérieur à celui contre les RREP. Ceci est dû à la différence de dissémination de messages RREQ et RREP. Dans ce qui suit, nous nous intéressons à la détection d'attaques complexes.

3.6.1.2 Résultats concernant les attaques complexes.

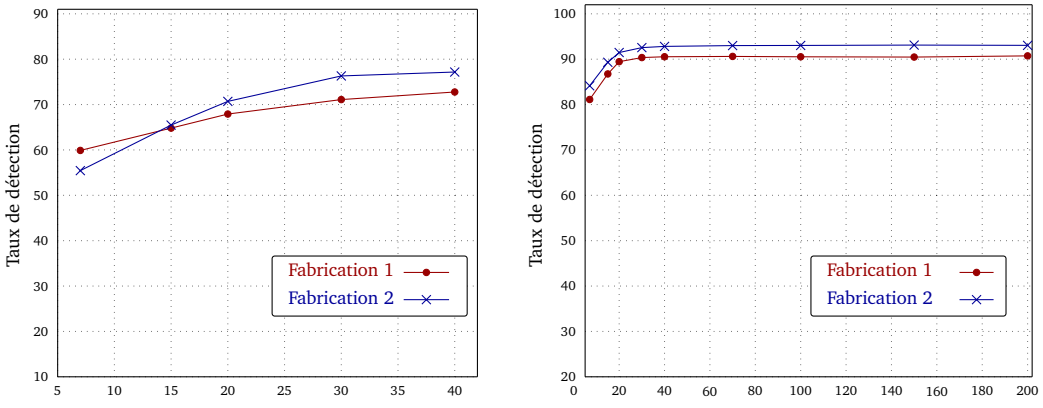
La seconde partie de nos expérimentations a porté sur les attaques permettant à un nœud malhonnête de s'insérer dans le maximum de chemins. Nous avons d'abord considéré l'effet produit par le changement de la densité du réseau sur le taux de détection. Ainsi, nous varions le nombre de nœuds de 20 à 200 sans varier la densité du trafic (même nombre de pairs émetteurs/récepteurs communiquant).



a. Fabrication 1 b. Fabrication 2

FIGURE 3.7 – Impact de la variation de la densité du réseau (attaques complexes)

Dans les figures 3.7a et 3.7b, nous notons que le taux de détection est relativement faible pour les réseaux de faibles densité de population et augmente considérablement en augmentant cette densité. Ceci est dû au fait que plus le réseau est dense, plus le nœud malhonnête a des voisins susceptibles de détecter son comportement. De nouveau, les cas de non-détection sont essentiellement dus au fait que ces attaques reposent exclusivement sur les réponses de routes RREP qui sont moins disséminées dans le réseau par rapport aux demandes de route RREQ.

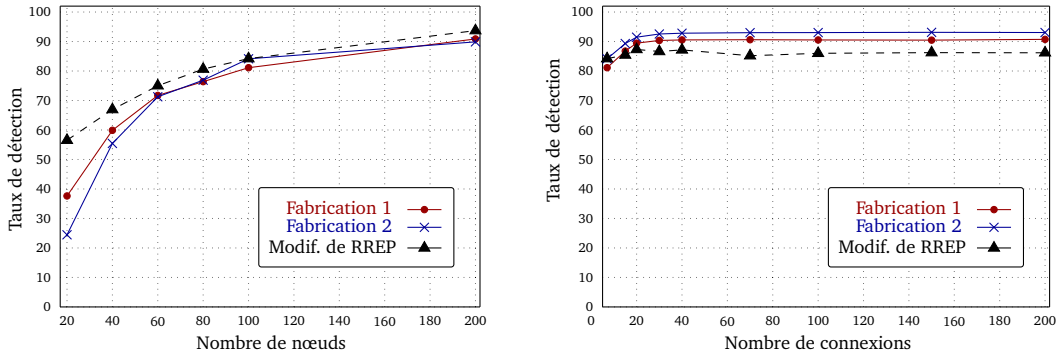


a. Réseau de 40 nœuds

b. Réseau de 100 nœuds

FIGURE 3.8 – Impact de la variation de la densité du trafic (attaques complexes)

Ensuite, nous nous intéressons à l'impact de la variation de la densité du trafic. Nous expérimentons la performance de notre système de détection dans des situations où le trafic est dense avec deux tailles de réseau : 40 nœuds (figure 3.8a) et 100 nœuds (figure 3.8b). Nous constatons que plus le trafic est important, plus le taux de détection augmente. Celui-ci est proche de 90% dans les réseaux à forte densité et ayant le plus de trafic. Ce phénomène est en fait logique : l'attaquant cherchant à se placer sur le plus de routes possibles effectue plus d'attaques lorsque le nombre de routes légitimes créées augmente, ce qui augmente les risques qu'il soit détecté. Parallèlement, plus de nœuds entendront ces attaques lorsque la densité du réseau augmente.



a. Variation de la densité du réseau

b. Variation de la densité du trafic dans un réseau de 100 nœuds

FIGURE 3.9 – Attaques complexes vs attaques élémentaires

Les résultats obtenus lors de l'évaluation des attaques complexes confirment ceux obtenus lors de l'évaluation des attaques élémentaires. Pour illustrer ces propos, la figure 3.9 compare entre les attaques complexes et l'attaque élémentaire par modification de RREP étant donnée que cette attaque élémentaire est la base de ces attaques complexes. Cette figure montre l'équivalence des résultats : alors que la variation de la densité du réseau montre des résultats équivalent surtout pour les grandes densités, la variation de la densité du trafic montre des taux de détection pour les attaques complexes supérieurs à ceux des attaques élémentaires.

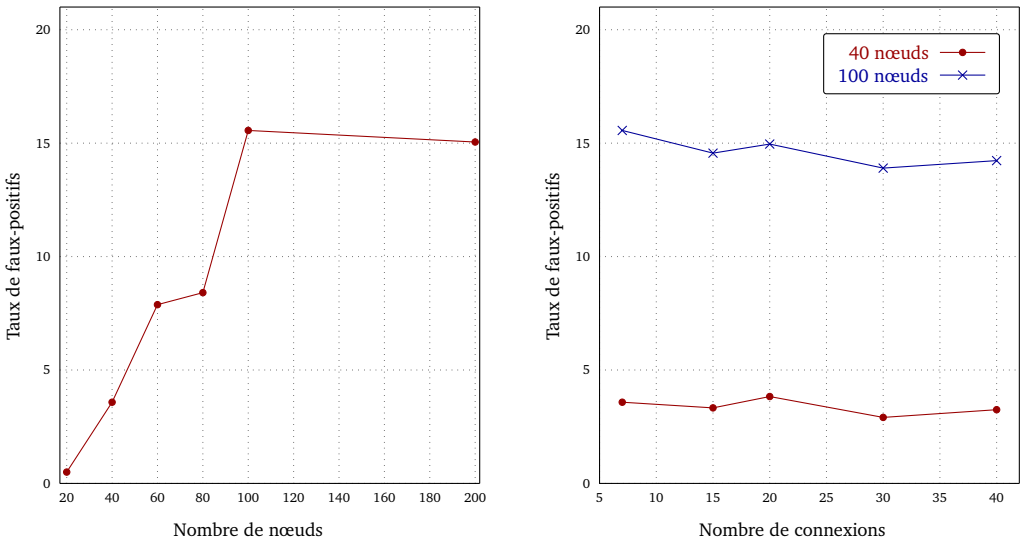
Dans ce qui suit, nous discutons les cas de faux positifs constatés.

3.6.1.3 Discussion concernant les faux-positifs.

Au cours de l'analyse des simulations précédemment effectuées, nous avons noté des cas de faux-positifs : il s'agit des nœuds qui sont placés dans la liste des suspects (*suspect list*) alors qu'ils sont honnêtes. Nous calculons ainsi le taux de faux-positifs (TFP) qui correspond au nombre de nœuds qui apparaissent au moins une fois dans la liste des suspects d'au moins un nœud honnête sur le total des nœuds honnêtes.

$$TFP = \frac{\sum noeuds\ honnêtes\ qui\ apparaissent\ dans\ la\ liste\ des\ suspects}{total\ des\ noeuds\ honnêtes} \times 100$$

Les résultats des simulations montrent que ce taux est négligeable (entre 0% et 1%) dans la majorité des attaques sauf pour l'attaque par modification du `#SNS` dans une RREQ où il peut atteindre 15% dans un réseau à forte densité de population et de trafic.



a. Variation de la densité du réseau b. Variation de la densité du trafic

FIGURE 3.10 – Taux de faux-positifs pour l'attaque par modification du `#SNS` dans une RREQ

La figure 3.10 présente la variation du taux de faux-positifs obtenus pour l'attaque par modification du `#SND` dans une RREQ. Il s'agit de la seule attaque où nous avons eu d'aussi grands taux.

Dans la figure 3.10a, la variation de la densité du réseau provoque l'augmentation du taux de faux positifs, alors que la variation de la densité du trafic ne semble pas avoir un effet remarquable (voir figure 3.10b) les taux se stabilisant autour de 4% et 15 % pour des réseaux de 40 et 100 nœuds respectivement. Une approximation de cette valeur est donnée dans la figure 3.10a pour chaque densité de population.

Nous avons identifié plusieurs causes à ces faux positifs :

– Le partitionnement du réseau peut induire les nœuds en erreur : certains nœuds suspectent de manière erronée des nœuds innocents qui viennent d'entrer dans leur voisinage. En effet, ces nœuds se déplacent d'une partition à une autre ce qui fait qu'ils n'ont pas la connaissance nécessaire pour juger de la culpabilité ou de l'innocence des nœuds formant leur nouveau voisinage. Étant donné que ces nouveaux arrivants sont honnêtes, ils se comportent normalement et retransmettent les paquets reçus, y compris ceux de l'attaquant, au risque d'être eux même détectés comme malhonnêtes.

– Dans une moindre mesure, les ré-émissions dues aux collisions peuvent dans certaines situations être détectées incorrectement comme étant des rejeux. Remarquons que ces collisions sont plus nombreuses dans des réseaux denses.

Il est important de noter que durant les expérimentations, les nœuds détectés et ajoutés à la liste des suspects ont été bannis pour une courte période de temps par les nœuds les ayant détectés afin de ne pas perturber le déroulement normal du protocole.

L'analyse de performance de notre système de détection montre sa fiabilité. En effet, les résultats obtenus lors des expérimentations présentent un grand taux de détection d'actions malhonnêtes avec un faible taux de faux positifs. Ceci nous amène dans un second temps à vérifier si le mécanisme que nous avons introduit n'influence pas les performances du protocole. Nous présentons cette analyse dans la section suivante.

3.6.2 Performance du protocole

Pour pouvoir analyser les performances du protocole, nous nous basons sur le calcul de métriques [CM99] pour montrer que les performances obtenues par le protocole auquel nous avons ajouté notre système de détection sont au moins équivalentes aux performances obtenues par le protocole de base (sans modification). Ainsi, pour chaque métrique, nous effectuons les mesures dans chacun des cas suivants :

– **Cas 1.** Exécution du protocole AODV sans aucune modification et sans attaque. Les courbes correspondantes sont notées « *AODV normal sans attaque* » et sont représentées par des courbes en trait plein rouge avec de gros points comme marqueurs.

– **Cas 2.** Exécution du protocole AODV sans aucune modification et avec attaque. Les courbes correspondantes sont notées « *AODV normal avec attaque* » et sont représentées par des courbes en pointillé noir avec de gros points comme marqueurs.

– **Cas 3.** Exécution du protocole AODV auquel nous avons ajouté notre système de détection sans attaque. Les courbes correspondantes sont notées « *AODV modifié sans attaque* » et sont représentées par des courbes en trait plein bleu avec des croix comme marqueurs.

– **Cas 4.** Exécution du protocole AODV auquel nous avons ajouté notre système de détection

avec attaque. Les courbes correspondantes sont notées « *AODV modifié avec attaque* » et sont représentées par des courbes en trait interrompu vert avec des croix comme marqueurs.

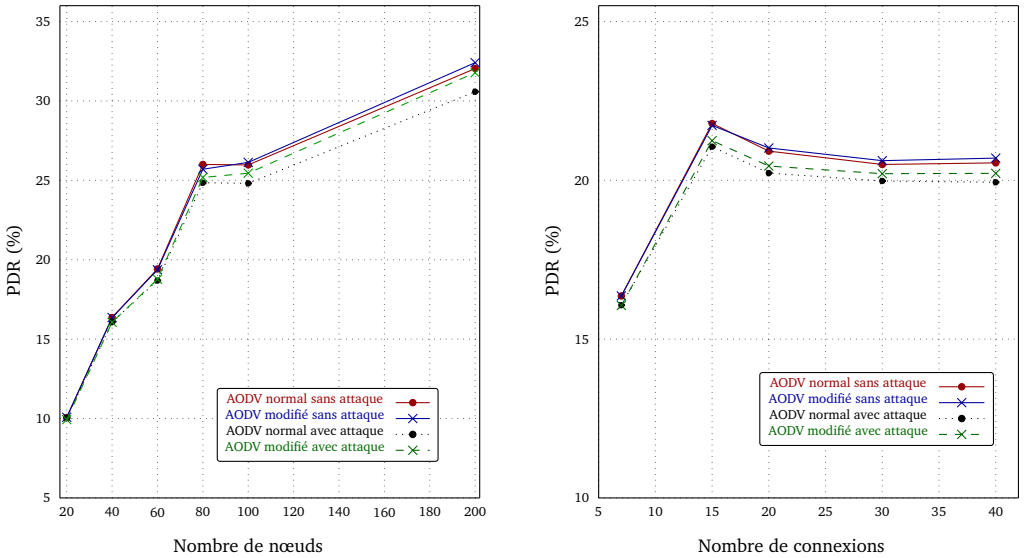
La comparaison des courbes obtenues à l'issue de l'exécution du cas 1 et 2, montre que l'ajout de notre système de détection n'influe pas sur les performances du protocole de base qui restent les mêmes et présente dans certains cas une légère amélioration dû aux optimisations apportées lors de l'implémentation. De même, la comparaison des courbes obtenues à l'issue de l'exécution du cas 3 et 4, montre qu'en cas d'attaques, l'extension que nous proposons présente de meilleurs performances par rapport au protocole de base tout en rendant le routage plus sûr. Nous définissons dans ce qui suit ces métriques et nous interprétons les résultats obtenus.

3.6.2.1 Taux de paquets reçus avec succès

Ce taux noté PDR pour *Packet Delivery Ratio* [Fri07] est le nombre de paquets de données reçus avec succès par la destination par rapport au nombre de paquets de données émis par la source. Il nous permet de vérifier si l'extension du protocole à un impact sur le transfert de paquets de données avec succès.

$$PDR = \frac{\textit{Nombre de paquets de données reçus par la Destination}}{\textit{Nombre de paquets de données émis par la Source}} \times 100$$

La figure 3.11 présente la variation de ce taux par rapport à la densité du réseau (figure 3.11a) et par rapport à la densité du trafic (figure 3.11b). La variation du PDR pour l'exécution d'AODV normal et l'exécution d'AODV modifié sans attaque sont identiques et se superposent ce qui montre que notre mécanisme ne dégrade pas la qualité du protocole en termes de proportions de paquets de données reçus par rapport au données émis.



a. Variation de la densité du réseau

b. Variation de la densité du trafic

FIGURE 3.11 – Taux de paquets émis/reçus avec succès (PDR)

La variation du PDR pour l'exécution d'AODV normal avec attaque montre les plus faibles taux de paquets délivrés avec succès. Ceci est logique puisque dans ces expérimentations nous avons mis en place des attaquants perturbant les opérations de routage ce qui explique la dégradation de ce taux. La variation du PDR pour l'exécution d'AODV modifié avec attaque montre un taux de paquets délivrés avec succès supérieur. Cependant, elle reste en dessous des performances du protocole de base sans attaques : ceci s'explique par le fait qu'en cas d'attaques, notre mécanisme détecte des comportements malhonnêtes et bloque ses initiateurs ce qui engendre un délai plus long pour l'établissement des chemins occasionnant une légère dégradation des performances.

3.6.2.2 Délai de bout-en-bout

Il s'agit du temps moyen qu'un paquet de données envoyé avec succès prend pour atteindre la destination. Il est dénoté RAL pour *Route Acquisition Latency* ou encore *End-to-End Delay* [Fri07]. Ce temps inclut le délai de traitement ainsi que le délai d'attente dans les files d'attentes dans chaque nœud intermédiaire. Il est calculé selon la formule suivante :

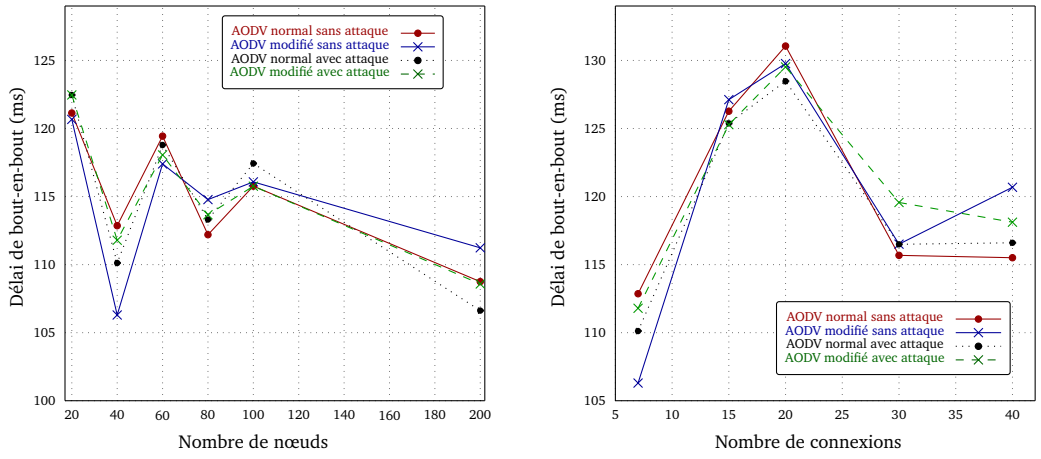
$$RAL = \frac{\sum [T_r(i) - T_e(i)]}{Nombre\,de\,paquets\,émis} \times 1000\,(ms)$$

avec :

$T_e(i)$: instant où le paquet « i » quitte la source.

$T_r(i)$: instant où le paquet « i » arrive à la destination.

La figure 3.12 montre l'évolution de ce temps en fonction de la densité du réseau et la densité du trafic. Nous remarquons l'alternance des valeurs obtenues : parfois une dégradation est constatée, d'autres fois notre proposition dégage une amélioration. Cependant, la différence observée entre la plus petite valeur et la plus grande reste très faible ($\approx$ 2 ou 3 ms). Ceci montre que l'implémentation du système de détection que nous proposons n'influe pas sur le traitement des paquets et n'engendre pas de temps de traitement supplémentaires ralentissant le déroulement normal du protocole et peut même l'améliorer dans certains cas.



a. Variation de la densité du réseau

b. Variation de la densité du trafic

FIGURE 3.12 – Délai de bout-en-bout

Certains résultats sont surprenant comme c'est le cas dans un réseau exécutant AODV sans modification incluant un attaquant : son action malhonnête semble diminuer le délai de bout en bout qui logiquement devrait augmenter. Ceci revient essentiellement au fait que l'attaquant continue à participer normalement aux opérations de routage après avoir exécuté son action malhonnête et il se peut que dans certains cas, il propose de meilleurs alternatives ce qui contribue à diminuer le délai de bout en bout.

3.6.2.3 Charge de routage

Nous évaluons cette métrique de deux manières différentes :

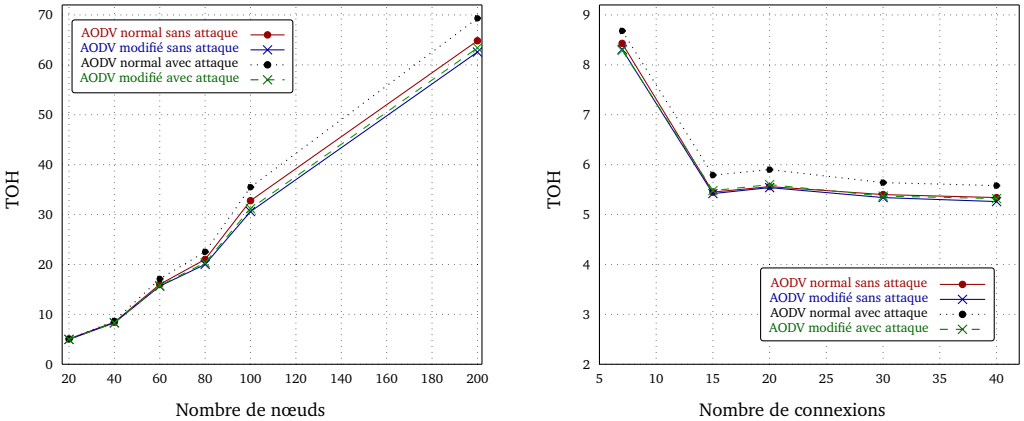
– Nous calculons le nombre de paquets de contrôle générés (*Traffic Overhead (TOH)* [Fri07]) : il s'agit de la proportion moyenne de paquets de routage pour chaque paquet de données envoyé.

$$TOH = \frac{\sum paquets\,de\,routage\,transmis}{\sum paquets\,de\,données\,délivrés\,à\,la\,Destination\,avec\,succès}$$

Cette métrique nous permet de donner un ordre de grandeur du nombre de paquets de routage moyen nécessaire pour envoyer un paquet de données. La figure 3.13 montre l'évolution de cette proportion :

– Dans un premier temps, nous varions la densité du réseau (figure 3.13a). Nous remarquons que jusqu'à une densité de 133 nœud/km^2 les courbes se superposent. Pour les densités supérieures, nous observons une légère baisse du nombre de paquets de contrôle générés pour les courbes mettant en place notre système de détection.

– Dans un second temps, nous varions la densité du trafic (figure 3.13b). Les courbes obtenues se superposent.



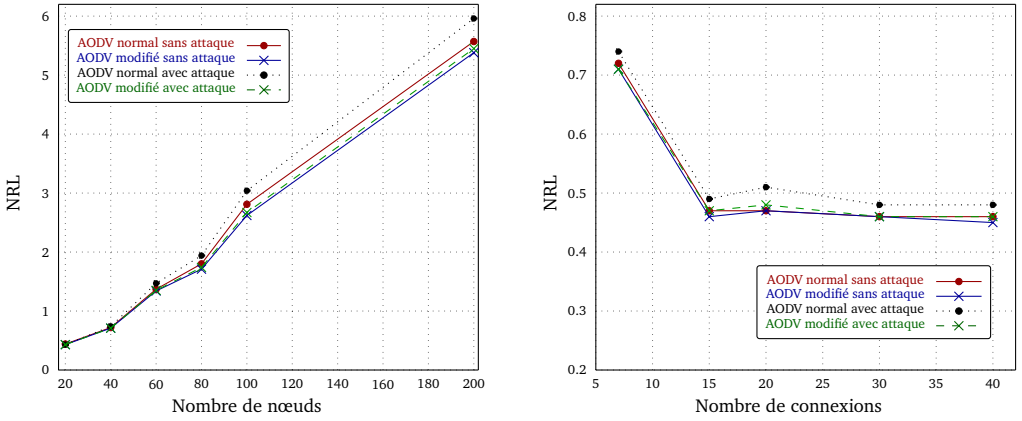
a. Variation de la densité du réseau

b. Variation de la densité du trafic

FIGURE 3.13 – Proportion des paquet de routage par rapport aux paquets de données (TOH)

– Nous calculons la charge de routage normalisée (*Normalized Routing Load (NRL)* [Fri07]) : Ce paramètre donne une idée sur la consommation de la bande passante par les paquets de contrôle par rapport aux paquets utiles de données.

$$NRL = \frac{\sum Nombre\,d'octets\,de\,paquets\,de\,routage\,transmis}{\sum Nombre\,d'octets\,de\,paquets\,de\,données\,délivrés\,avec\,succès\,à\,la\,destination}$$



a. Variation de la densité du réseau

b. Variation de la densité du trafic

FIGURE 3.14 – Charge de routage normalisée (NRL)

En analysant les résultats (figure 3.14), nous obtenons les mêmes courbes que pour le TOH. La différence qui existe entre ces deux métriques c'est que le TOH donne une idée sur le nombre de paquets de routage générés pour chaque paquet de données alors que le NRL donne une idée sur le nombre de bits de routage nécessaires par bit de données envoyé avec succès ce qui nous donne une idée sur la bande passante utile.

Comme nous pouvons le constater à travers ces expérimentations, l'extension que nous avons introduit n'influe pas sur les performances générales du protocole, elle l'améliore même dans certains cas. Cependant, les courbes obtenues pour chaque paramètre étudié ne présentent pas des écarts significatifs par rapport au cas d'un réseau exécutant AODV de base avec attaque ce qui peut prêter à douter de l'efficacité des attaques proposées. En effet, un nœud malhonnête tel que nous l'avons modélisé intervient seulement lors du processus de création de route et participe ensuite normalement aux opérations d'échange de paquets de données. Ces actions malveillantes se traduisent en une légère variation du nombre de paquets de routage échangés. Et comme les métriques utilisées pour l'évaluation des performances du protocole se basent essentiellement sur les paquets de données, nous obtenons des écarts non significatifs entre les courbes.

Dans ce qui suit, nous proposons d'étendre le comportement malhonnête de telle sorte qu'il touche aussi les paquets de données. Après exécution de l'attaque lui permettant de s'insérer sur une route, le nœud malhonnête ne transmet pas les paquets de données : il se contente de les effacer. Cette modification aura un impact direct sur les performances du protocole. Nous expérimentons ainsi ce nouveau type de comportement malhonnête.

La figure 3.15 présente l'évolution du taux de paquets délivrés avec succès dans les cas où les nœuds exécutent : (1) AODV de base sans attaque, (2) AODV de base avec attaque, et (3) AODV modifié avec attaque. La courbe ainsi obtenue avec l'extension que nous avons proposé montre l'amélioration du taux de paquets délivrés avec succès par rapport à un réseau exécutant AODV

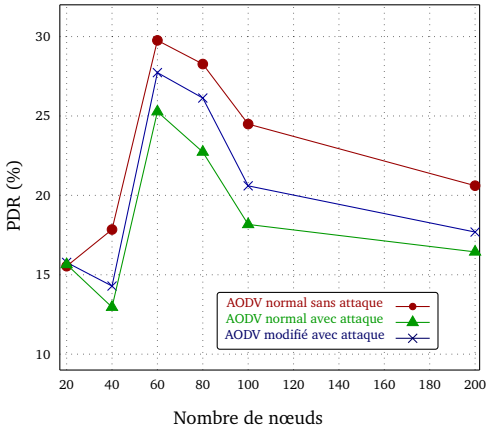
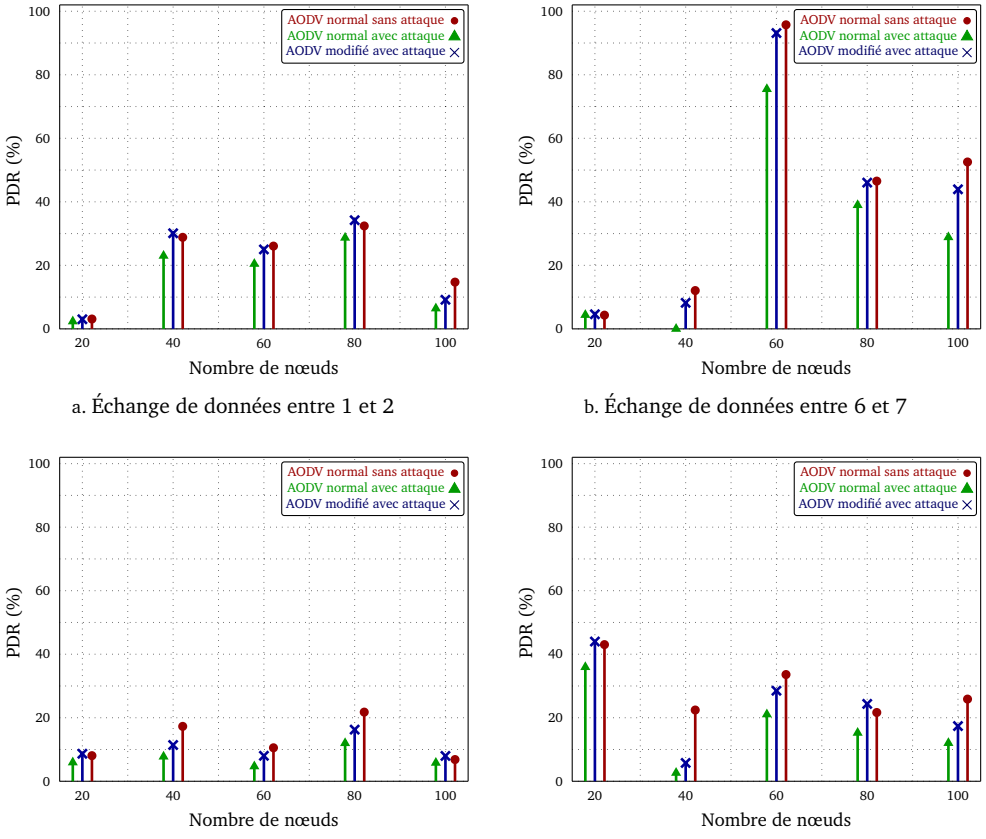


FIGURE 3.15 – Taux de paquets délivrés avec succès pour l'attaque sur les paquets de données

de base dans les mêmes circonstances (nombre d'attaquants, disposition des nœuds, scénario de mouvement, scénario de mobilité). Toutefois, cette différence n'est pas significative puisqu'elle constitue la moyenne des différents échanges entre les couples d'émetteurs/récepteurs.



a. Échange de données entre 1 et 2

b. Échange de données entre 6 et 7

c. Échange de données entre 8 et 9

d. Échange de données entre 14 et 15

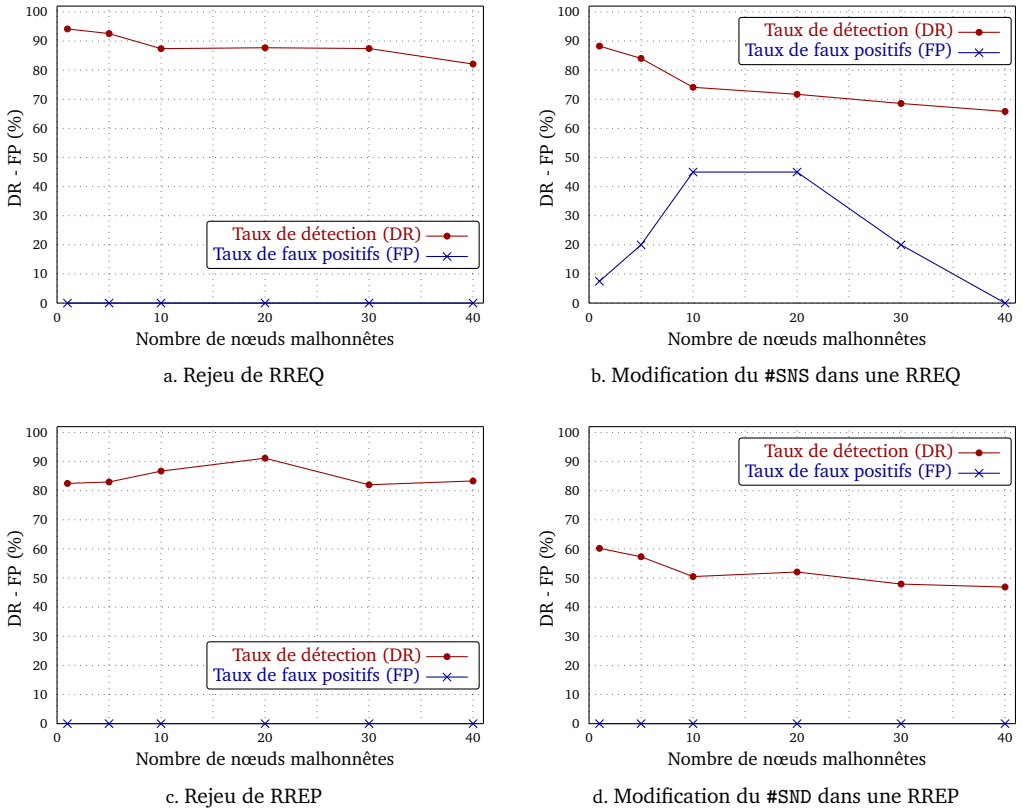
FIGURE 3.16 – Taux de paquets délivrés avec succès pour certains couples émetteur/récepteur

La figure 3.16 présente le taux de paquets délivrés avec succès pour 4 couples de nœuds communiquant. Comme nous pouvons le constater, l'effet produit par l'attaquant diffère d'un chemin à un autre et d'une densité de réseau à une autre. Ceci est dû au fait que l'attaquant se déplace et ne cible pas une communication en particulier ce qui explique un PDR non nul dans la majorité des cas. En observant les histogrammes, nous pouvons affirmer que l'extension que nous proposons (barres du milieu se terminant par une croix) améliore les performances du protocole de base en cas d'attaques (barres se terminant par un triangle) et se rapproche des taux obtenus avec le protocole de base sans attaques (barres se terminant par un cercle plein). Ceci met en évidence l'avantage de notre extension puisqu'elle créée une ligne de défense rendant le protocole AODV plus robuste envers les attaques menées par les nœuds malveillants.

3.6.3 Mise à l'épreuve du système de détection

Nous procédons dans cette section au test du comportement de notre système de détection dans des cas où le nombre d'attaquants varie. Nous utilisons à cet effet le taux de détection d'action malhonnêtes précédemment introduit.

La figure 3.17 représente l'évolution du taux de détection et du taux de faux positifs en variant le pourcentage de nœuds malhonnêtes dans un réseau de 40 nœuds. Le trafic entre les nœuds reste constant (7 paires de nœuds émetteurs/récepteurs).



a. Rejeu de RREQ

b. Modification du `#SNS` dans une RREQ

c. Rejeu de RREP

d. Modification du `#SND` dans une RREP

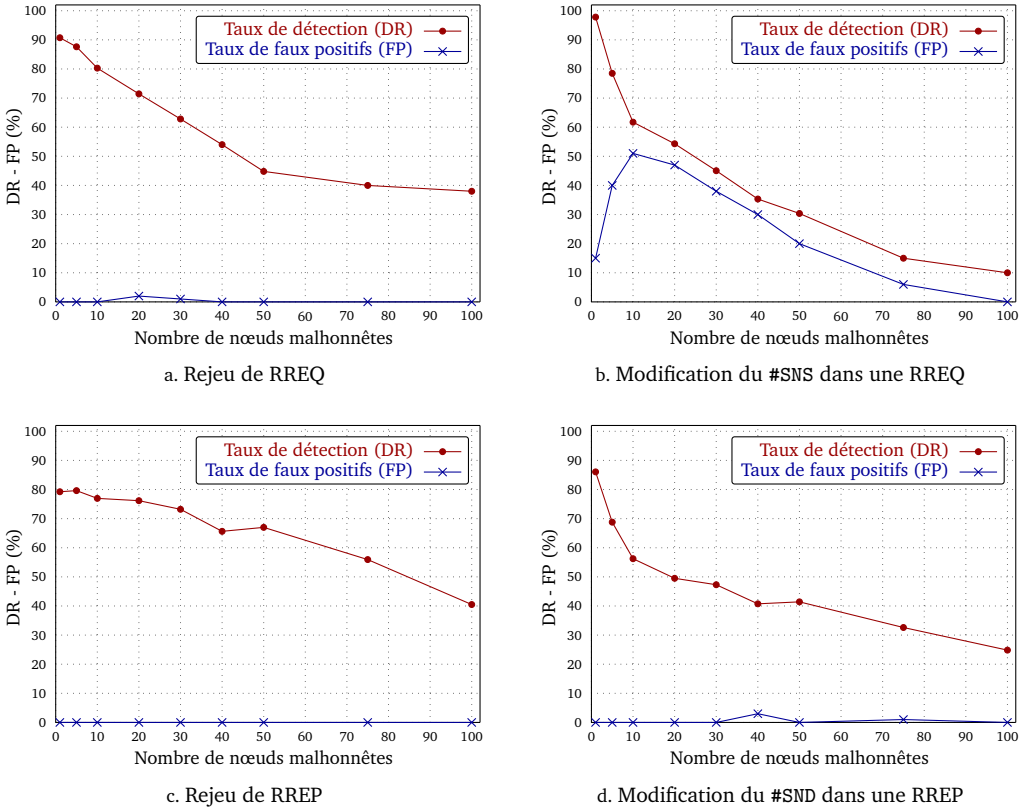
FIGURE 3.17 – Impact de la variation du nombre d'attaquants pour un réseau de 40 nœuds

Nous remarquons que le taux de détection ne baisse que légèrement en augmentant le nombre de nœuds malhonnêtes perturbant le bon déroulement des actions dans le réseau. Ceci montre l'efficacité du système de détection face à l'augmentation du nombre d'attaquants dans le réseau.

La courbe présentant le taux de faux positifs varie d'une attaque à une autre :

– Ce taux reste nul pour l'attaque par rejeu de demande de route (figure 3.17a) et l'attaque par rejeu ou modification de réponse de route (respectivement les figures 3.17c et 3.17d) malgré l'augmentation du nombre d'attaquants.

– Comme nous avons pu le constater dans le paragraphe discutant les cas de faux positifs (page 76), le taux des nœuds détectés par erreur comme étant malhonnêtes alors qu'ils ne le sont pas pour l'attaque par modification de la RREQ varie en augmentant la densité du réseau. Il en est de même en augmentant le nombre d'attaquants dans un réseau de taille fixe (figure 3.17b). Il est intéressant de constater que cette courbe atteint un pic (entre 25% et 50% de nœuds malhonnêtes) avant de décroitre et s'annuler. Ceci est dû au fait que plus le nombre d'attaquant est grand, plus le nombre de nœuds malhonnêtes qui seront réellement détectés augmente ce qui diminue le nombre de nœuds considérés malhonnêtes par erreur.



a. Rejeu de RREQ

b. Modification du `#SNS` dans une RREQ

c. Rejeu de RREP

d. Modification du `#SND` dans une RREP

FIGURE 3.18 – Impact de la variation du nombre d'attaquants pour un réseau de 100 nœuds

Les courbes de la figure 3.18 représentent l'évolution du taux de détection et du taux de faux positifs en variant le pourcentage de nœuds malhonnêtes dans un réseau de 100 nœuds. Ils confirment les résultats obtenus avec un réseau de 40 nœuds. La tendance générale qui se dégage de ces résultats est la dégradation plus au moins importante du taux de détection en augmentant

le nombre de nœuds malhonnêtes opérant dans le réseau. Parallèlement, le taux de faux positifs a tendance à augmenter.

Nous constatons que le comportement du système de détection varie avec la taille du réseau : pour un réseau de faible densité (par exemple 20 ou 40 nœuds), la variation du nombre de nœuds malhonnêtes n'a pas d'effet significatif sur le système alors que dans un réseau plus grand (100 nœuds par exemple) la dégradation des performances est plus accentuée. Ceci est dû au partitionnement du réseau qui est plus important pour les plus faibles densités où le nombre d'attaquants appartenant à la même partition reste restreint (même s'il y en a d'autres en dehors de la partition) ce qui explique les taux de détection élevés pour ces réseaux alors que ce n'est plus le cas dans les réseaux à forte densité. Dans ces cas, nous jugeons que le système de détection n'est plus fiable au delà de 25% de nœuds malhonnêtes dans le réseau puisque le taux de détection descend en deçà de 60%.

3.7 Résumé

Tout au long de ce chapitre, nous avons présenté les détails de l'implémentation qui nous ont permis de mettre en place des simulations pour tester la solution. Ainsi, après la présentation de l'environnement de simulation, nous avons détaillé le fonctionnement de l'implémentation d'AODV choisie à travers la description du cheminement des paquets entre les fonctions. Ceci nous a permis de retrouver l'endroit où ajouter les contrôles (implémentant les critères d'incohérences) permettant de détecter les comportements malhonnêtes.

Afin de lancer les simulations, il fallait mettre en place les comportements malhonnêtes. Nous avons décrit un ensemble d'attaques élémentaires et d'autres composées basées sur des attaques élémentaires.

L'analyse des résultats obtenus après l'exécution des simulations a porté sur trois axes :

- Le premier concerne l'efficacité de notre système de détection. Nous calculons le taux de détection en déterminant le nombre de voisins détectant une attaque par rapport à la totalité des voisins. Les résultats obtenus permettent d'affirmer l'efficacité du système de détection que nous avons mis en place.
- Le second montre que l'extension que nous proposons ne dégrade pas les performances du protocole. À travers le calcul de métriques tels que le taux de paquets reçus avec succès, le délai de bout-en-bout, ou la charge de routage, nous montrons que l'exécution du protocole étendu (même en cas d'attaques) a des performances semblables voire meilleures dans certains cas que le protocole de base.
- Le troisième teste les limites de la solution dans un environnement fortement corrompu. Ainsi, nous observons que la dégradation de la détection est dépendante de la densité du réseau et que les faux-positifs augmentent avec le nombre d'attaquants.

Ainsi, nous démontrons que le système de détection d'actions malhonnêtes est efficace sans pour autant toucher aux performances du protocole : il permet d'ajouter une ligne de défense supplémentaire au protocole.

Dans la suite, nous nous intéressons à la réduction des faux positifs. Nous modifions le raisonnement pour agir sur les règles et limiter les fausses accusations. Nous décrivons la différence par rapport au système actuel et nous expérimentons ce système pour vérifier ces performances.

CHAPITRE 4

ÉTUDE DES FAUX-POSITIFS

Suite aux taux de faux positifs dans certaines expérimentations, nous nous intéressons dans ce chapitre de plus prêt à ce problème pour diminuer ce taux voir l'annuler ; l'origine de ce problème étant la mauvaise interprétation d'un critère d'incohérence suite à des circonstances pouvant prêter à confusion et induisant en erreur la prise de décision.

En se basant sur les critères que nous avons déjà dégagé, nous proposons de les revisiter pour proposer de nouveaux critères permettant de prendre une décision *ferme* : le nœud traitant le message peut affirmer sans aucun doute que le nœud initiant le message est un nœud malhonnête. Cette approche doit se fier à des sources fiables lors du recoupement des messages pour ne pas accuser à tort un nœud honnête. Nous étudions, implémentons et expérimentons cette approche dans la section 4.1.

Cette approche que nous notons approche 1 étant plus restrictive impliquant une baisse des taux de détection, nous proposons dans la section 4.2 un complément de détection se basant sur l'avis de la majorité dans les situations où une décision *ferme* n'est pas possible. Nous détaillons ainsi cette approche et nous présentons les résultats expérimentaux obtenus. Nous discutons enfin les mesures prises à l'encontre des nœuds malveillants dans la section 4.3. Nous montrons que les performances de détection peuvent être améliorées en jouant sur les mesures de punition.

4.1 Approche 1

Pour éviter de prendre de mauvaises décisions dans certaines situations, nous proposons de se fier seulement aux critères permettant d'accuser sans aucun doute un nœud de malhonnêteté. Le nœud traitant possédant ainsi la preuve irréfutable de la culpabilité d'un autre nœud, le bannit et ne traite plus les messages venant de lui.

Concernant les attaques par rejeu de demande de route et par rejeu ou modification de réponse de route, les critères d'incohérence dégagés dans le chapitre 2 ne changent pas puisque ceux-ci permettent d'avoir un taux de détection d'actions malhonnêtes important sans pour autant générer de fausses accusations (exception faite de certains cas très rares).

L'attaque par modification de demande de route est l'attaque où les cas de faux positifs apparaissent le plus souvent. Ceci est dû essentiellement au partitionnement du réseau et au changement brusque du voisinage de certains nœuds qui sont alors accusés de manière erronée. Les nœuds n'ont souvent pas une connaissance suffisamment complète pour décider. Alors, en se comportant normalement (en relayant les messages des autres nœuds, y compris ceux des attaquants), ils risquent d'être eux-mêmes détectés comme malhonnêtes et de propager l'attaque.

Nous proposons dans ce qui suit de revisiter le critère d'incohérence concerné (critère d'incohérence 2.4) afin de ne garder que les cas où la décision d'accusation ne peut être réfutée.

4.1.1 Modifications apportées au critère d'incohérence

La détection des nœuds malveillants ainsi que les fausses accusations sont générées par le critère d'incohérence 2.4 pour l'attaque par modification d'une demande de route. Nous rappelons ce critère pour pouvoir expliquer par la suite les améliorations apportées.

Ainsi, à la réception d'une demande de route ($M \xleftarrow{\text{RREQ}_N}$), qui a été déjà reçue et traitée ($isTreated(RREQ_N, THE_M) = True$) mais qui est reçue pour la première fois du voisin N ($THE_M.RREQ_N.ip\_src \neq N$), M compare le numéro de séquence de la source reçu dans la RREQ ($RREQ_N.\#SNS$) avec celui trouvé dans sa table de routage ($RT_M.(RREQ_N.@Src).\#SNS$). S'il les trouve différents, il en déduit que cette demande de route a été modifiée.

L'expression 2.4 présente le critère d'incohérence introduisant la méfiance qu'il faut développer à l'envers d'un nœud se comportant de la sorte.

Critère d'incohérence 2.4

$$M \xleftarrow{\text{RREQ}_N} \wedge\ isTreated(RREQ_N, THE_M) = True \wedge\ THE_M.RREQ_N.ip\_src \neq N \wedge\ RREQ_N.\#SNS \neq RT_M.(RREQ_N.@Src).\#SNS \Rightarrow\ M\ \neg trusts\ (N)$$

Chapitre 2 - Analyse des aspects de confiance dans AODV, page 57.

Il s'agit de recouper les informations fraichement reçues avec celles qui sont stockées dans la table d'historique étendue (THE) afin de déceler les incohérences. Pour le critère 2.4 (rappelé ci-dessus), l'incohérence réside dans le fait d'avoir un numéro de séquence de la source reçu dans le message RREQ différent de celui qu'il a auparavant reçu d'un autre nœud alors qu'il s'agit de la même demande de route. Ceci peut prêter à confusion dans certains cas où la demande de route de l'attaquant atteint en premier le nœud cible ce qui fait que celle-ci devient sa référence dans sa table d'historique et ainsi il détecte toute autre demande de route qui diffère comme provenant d'un nœud malicieux même si celle-ci est légitime.

Le problème avec ce critère d'incohérence réside dans le choix de la référence prise dans la table d'historique étendue sur laquelle le nœud se base pour estimer l'honnêteté de son voisin. Avec l'implémentation actuelle, le nœud traitant se base sur la première demande de route reçue. Nous proposons d'intervenir à ce niveau en réglementant le choix, dans la THE, du message sur lequel le nœud se base pour prendre sa décision. Ainsi, nous proposons de préciser les cas où un nœud peut disposer d'informations fiables ce qui lui permettra de prendre une décision sûre sans être induit en erreur.

Après analyse, nous dégageons les cas suivants :

– La source du message est évidemment le premier nœud qui peut affirmer de manière formelle une modification de message en entendant la retransmission d'un voisin :

$$M = RREQ_N.@Src$$

– Le voisin de la source d'un message peut aussi vérifier la retransmission de son voisin et déceler toute modification puisqu'il a reçu auparavant le message de la source qui l'a émis. Ainsi, la comparaison se fait par rapport au message reçu auparavant de cette source ce qui constitue une source fiable.

$$M \neq RREQ_N.@Src \ \wedge \ RT_M.(RREQ_N.@Src).\#HC = 1$$
$$\wedge \ THE_M.RREQ_N.ip\_src = RREQ_N.@Src$$

– Au delà (à deux sauts ou plus de la source), les nœuds n'ont plus d'informations sûres leurs permettant d'évaluer leur voisin. Plusieurs techniques peuvent alors être utilisées pour détecter d'autres cas mais comme nous allons le présenter ultérieurement (dans la section 4.2), le raisonnement adopté par la suite peut générer des cas de faux positifs. C'est pourquoi nous nous limitons au départ aux deux premiers cas.

Le critère d'incohérence 2.4 devient alors :

Critère d'incohérence 2.4'

$$M \xleftarrow{\text{RREQ}_N} \ \wedge \ isTreated(RREQ_N, THE_M) = True \ \wedge \ THE_M.RREQ_N.ip\_src \neq N$$
$$\wedge \left[\begin{array}{c} (M = RREQ_N.@Src) \\ \vee \\ \left(\begin{array}{c} M \neq RREQ_N.@Src \ \wedge \ RT_M.(RREQ_N.@Src).\#HC = 1 \\ \wedge \ THE_M.RREQ_N.ip\_src = RREQ_N.@Src \end{array} \right) \end{array} \right]$$
$$\wedge \ RREQ_N.\#SNS \neq RT_M.(RREQ_N.@Src).\#SNS$$
$$\Rightarrow \ M \neg trusts\,(N)$$

De cette manière, nous limitons les cas où un nœud peut décider de l'honnêteté de ses voisins. Cette limitation touche l'information qui peut être utilisée pour le recoupement d'information. Le choix de cette information dans la THE est soumis dorénavant à une vérification de fiabilité de la source. Toutefois, nous considérons que l'information est fiable que si elle provient directement de la source ou si le nœud traitant est lui même la source.

Dans ce qui suit, nous intégrons cette règle dans notre système afin de mesurer son impact sur le taux de détection ainsi que le taux de faux positifs.

4.1.2 Implémentation

Dans cette section, nous décrivons la mise en place du nouveau critère d'incohérence au système actuel. Toutefois, nous conservons l'ancienne implémentation de la règle de telle sorte à pouvoir simuler dans le même réseau des nœuds exécutant la version précédente ainsi que d'autres avec la nouvelle version. Ceci est déjà prévu par l'intermédiaire de la variable `reasoning_mode` indiquant le comportement du nœud lors de l'initialisation de la simulation. Ainsi, il suffit d'attribuer une nouvelle valeur non utilisée à cette variable qui correspondra au raisonnement incluant la nouvelle règle. Cette variable peut alors avoir les valeurs suivantes :

- 0 indique que le nœud ne procède à aucun raisonnement, il exécute AODV normal sans aucune modification ;
- 1 indique que le mode raisonnement de base est activé pour le nœud ;
- 2 indique que le mode raisonnement prenant en compte la nouvelle modification est activé pour le nœud.

Plusieurs changements ont été effectués au niveau de l'implémentation actuelle pour faire cohabiter le nouveau raisonnement avec le raisonnement de base. Ces modifications concernent essentiellement (i) les contrôles permettant l'exécution des règles, et (ii) l'ajout du nouveau critère encapsulé dans un contrôle vérifiant le type de raisonnement du nœud et qui doit être placé au même niveau que l'ancien critère. Nous présentons ci-après l'ajout à effectuer sous forme de pseudo-code.

```
Si (reasoning_mode == 2) Alors
    // Le nœud M reçoit/entend une RREQ du nœud N
    Si ( [(M est la source) or (M est voisin de la source
            and RREQ trouvée dans la THE provient de la source)]
                    and [#SNS reçu ≠ #SNS trouvé dans la THE] ) Alors
        N a modifié le numéro de séquence de la source dans la RREQ relayée
        Prendre les mesures nécessaire contre le nœud N
```

Une gestion de liste noire de nœuds malhonnêtes est ajoutée à l'implémentation. Ces nœuds ont été détectés par le critère d'incohérence que nous venons de présenter. Ce qui permet de punir ces nœuds en les bannissant du réseau : ceci correspond en fait à rejeter tout message provenant de ces nœuds. Le pseudo-code présenté ci-après doit être ajouté au début de la fonction `rreq_process` avant de commencer le traitement relatif à la RREQ ce qui permet de rejeter le paquet si son émetteur fait partie de cette liste.

```
Si (reasoning_mode == 2) Alors
    // Le nœud M reçoit/entend une RREQ du nœud N
    // Vérifier si le nœud duquel le message est reçu n'est pas dans
    la liste noire
    Si (N ∈ Liste_Noire) Alors
        Return ;
```

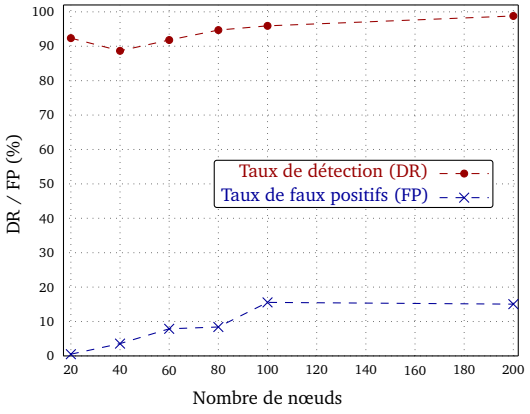
4.1.3 Résultats expérimentaux

Nous considérons les mêmes paramètres présentés dans la section 3.5 pour l'exécution des simulations. Nous procédons alors directement à l'étude des résultats obtenus.

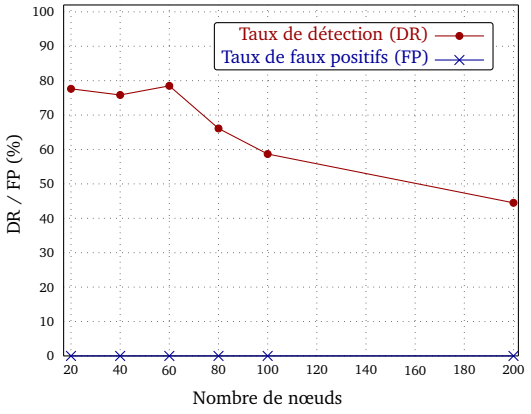
Nous évaluons le taux de détection d'actions malhonnêtes ainsi que le taux de faux-positifs en fonction de :

– **La densité du réseau** : nous varions le nombre de nœuds de 20 à 200 tout en gardant constant le trafic généré entre les nœuds (7 paires d'émetteurs/récepteurs). Le nombre d'attaquants ne change pas non plus et est fixé à un dans toutes les simulations.

  La figure 4.1b représente les résultats obtenus pour l'attaque par modification du numéro de séquence de la source (#SNS) dans une demande de route RREQ. Le système de détection utilisé implémente le critère d'incohérence 2.4' alors que la figure 4.1a représente les résultats obtenus pour la même attaque en utilisant le système de détection mettant en place le critère d'incohérence 2.4.



a. Système de détection mettant en place le critère d'incohérence 2.4



b. Système de détection mettant en place le critère d'incohérence 2.4'

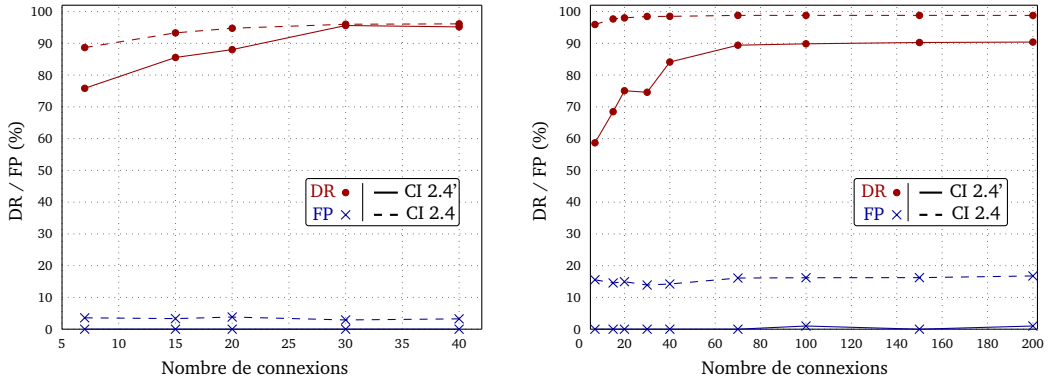
FIGURE 4.1 – Impact de la variation de la densité du réseau

Comme nous pouvons le constater, le taux de faux positifs reste nul malgré l'augmentation de la densité du réseau. Ce résultat est prévisible puisque nous nous basons sur le recoupement par rapport à des informations qui proviennent de la source elle même. Ce qui évite les fausses accusations. Toutefois, nous observons que les taux de détection sont inférieurs à

ceux obtenues avec la première version du critère. Pour les plus faibles densités, le taux de détection avoisine les 80% ensuite les valeurs affichent une dégradation pour les réseaux denses. Ceci est dû au faits que :

- Moins le réseau est dense, plus il est partitionné et les partitions sont de petites tailles. L'attaquant doit recevoir la demande de route pour qu'il puisse exécuter son action malhonnête (qui consiste à modifier le #SNS). Ceci n'est possible que s'il appartient à la même partition que la source. Et comme ces partitions sont de petites tailles, la probabilité qu'il soit à un ou deux sauts de la source est plus importante ce qui permet de détecter son action malhonnête. Les résultats obtenus après simulations montrent que dans $^4/_5$ des cas, le nœud malveillant est détecté. Ce pourcentage dépend aussi des mesures prises à l'encontre des nœuds malveillants comme nous le verrons plus tard dans la section 4.3.
- Plus le réseau est dense, moins il est partitionné et plus les partitions sont de grandes tailles, ce qui offre une connectivité plus importante. Ceci ouvre plus de possibilités de positionnement au nœud malhonnête : il peut se placer de plus en plus loin de la source (plus de 2 sauts) ce qui se traduit par une baisse progressive du taux de détection. De même, plus le réseau est dense, plus il y aura de voisins qui n'ont pas entendu le message émis par la source et ne détecteront pas l'attaque.

– **Densité du trafic** : Nous varions le nombre de connexions de 7 à 40 dans un réseau de 40 nœuds (figure 4.2a) et de 7 à 200 dans un réseau de 100 nœuds (figure 4.2b). Le nombre d'attaquants est le même dans les deux cas.



a. Réseau de 40 nœuds

b. Réseau de 100 nœuds

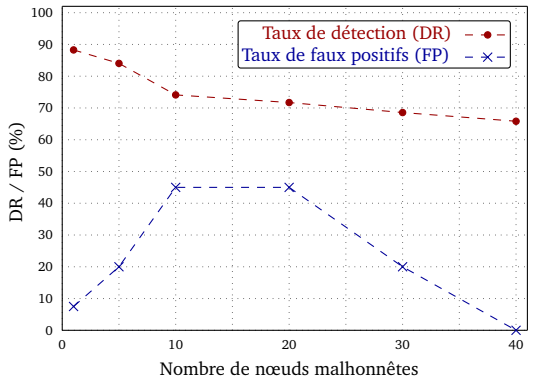
FIGURE 4.2 – Impact de la variation de la densité du trafic

Le système de détection intégrant le critère d'incohérence 2.4' ne génère pas de cas de faux positifs et propose des taux de détection d'actions malhonnêtes élevés (courbe à points en trait continu) qui se rapprochent de ceux obtenus avec l'ancien critère (courbe à points en trait interrompu) surtout dans des situations où le trafic est très dense.

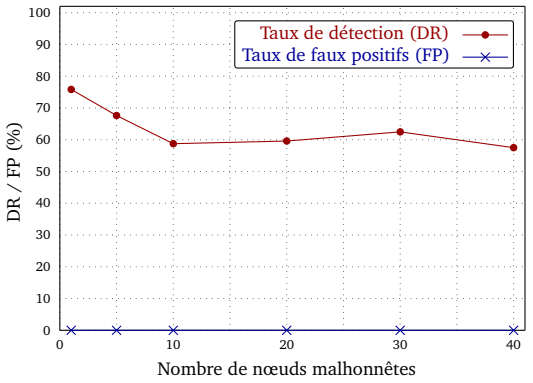
L'augmentation du taux de détection est dû au fait que plus le trafic est dense, plus les nœuds émetteurs sont dispersés dans le réseau augmentant ainsi la probabilité qu'ils soient à côté d'un attaquant. Le taux de détection dépend aussi des mesures prises à l'encontre des nœuds malveillants (voir section 4.3).

– **Nombre d'attaquants** : nous nous intéressons enfin à observer le comportement de notre système de détection dans des cas où le nombre d'attaquants augmente.

La figure 4.3b représente les résultats obtenus pour l'attaque par modification du #SNS pour une RREQ en utilisant le système de détection implémentant le critère d'incohérence 2.4' par rapport aux résultats obtenus pour la même attaque en utilisant le système de détection mettant en place le critère d'incohérence 2.4 (figure 4.3a).



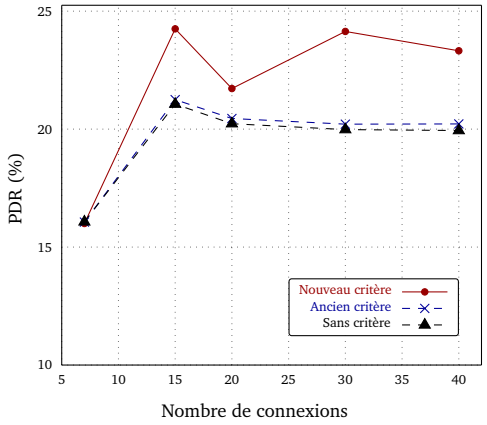
a. Système de détection mettant en place le critère d'incohérence 2.4



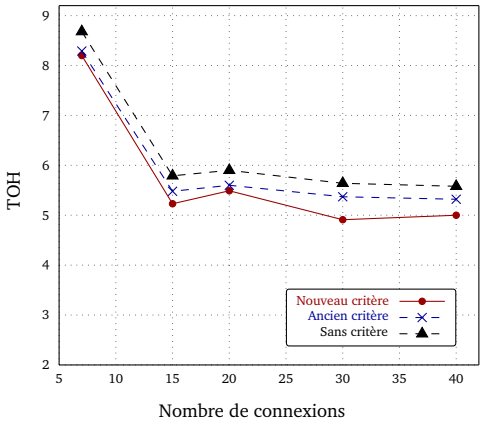
b. Système de détection mettant en place le critère d'incohérence 2.4'

FIGURE 4.3 – Impact de la variation du nombre d'attaquants dans un réseau de 40 nœuds

Comme nous pouvons le constater dans la figure 4.3b, le taux de faux positifs est nul malgré l'augmentation du nombre d'attaquants. Le taux de détection d'actions malhonnêtes est légèrement inférieur à celui obtenu avec le système mettant en place l'ancien critère d'incohérence.



a. Taux de paquets émis/reçus avec succès (PDR)



b. Proportion des paquet de routage par rapport aux paquets de données (TOH)

FIGURE 4.4 – Variation de la densité du trafic dans un réseau de 40 nœuds

La nouvelle version améliore aussi les performances du protocole puisque la différence de détection et l'absence de faux positifs font qu'il y a moins de nœuds bloqués par erreur. Ceux-ci

participent ainsi au routage ce qui améliore légèrement les performances du protocole surtout en variant le trafic dans le réseau. Les figures 4.4a et 4.4b montrent cette amélioration par rapport aux mesures effectuées pour les paramètres (PDR et TOH respectivement) dans un réseau utilisant l'ancien système de détection.

Suite à l'évaluation des performances du système de détection mettant en place le nouveau critère d'incohérence, nous obtenons de bonnes performances. Certes, elles sont légèrement moins bonnes que celles de la précédente version mais nous avons atteint l'objectif d'annuler les cas de fausses accusations. Dans ce qui suit, nous proposons de mettre en place un complément à l'approche 1 permettant d'améliorer la détection de nœuds malhonnêtes.

4.2 Approche 2 : complément de détection

4.2.1 Principe

L'idée se base sur le fait de se fier à l'avis de la majorité des voisins lors de la prise de décision. Le nœud traitant collecte ainsi les messages du voisinage et ne prend en considération que la requête dont la majorité des voisins sont d'accord dessus. Il rejette toute requête différente et accuse son émetteur de malhonnêteté.

Nous distinguons deux cas de figure :

Cas 1. Le nœud diffère le traitement de la demande de route et l'associe à la prise de décision. Ainsi, lors de l'exécution de la procédure de décision, le nœud distingue les nœuds honnêtes et déclenche à la suite le traitement normal de la demande de route provenant d'un nœud faisant partie de cet ensemble de nœuds honnêtes.

Cas 2. Le nœud traite normalement la première demande de route reçue et diffère la prise de décision. Ainsi, après avoir reçu toutes les demandes similaires parvenues du voisinage, il déclenche une procédure de décision qui lui permet de retrouver les nœuds malhonnêtes en se fiant à l'avis de la majorité.

Nous combinons cette approche à celle présenté dans la section 4.1 de telle sorte à couvrir les cas de figure qui ne sont pas traités par celle-ci. Rappelons que l'approche 1 repose sur le fait de ne prendre une décision que si le nœud est sûr de l'action malhonnête du voisin. Ainsi, un nœud ne prend de décision que si : i) il est la source du message de demande de route, ou ii) il est voisin de la source ayant initié le message de demande de route. Au delà, un nœud ne peut pas prendre de décision certaine. Nous proposons d'intervenir à ce niveau en mettant en place un système de prise de décision déclenché après avoir reçu la totalité des messages des voisins.

4.2.2 Mise en place

Nous décrivons dans ce qui suit l'intégration de cette solution à l'implémentation actuelle. Nous définissons ainsi de nouvelles valeurs pour la variable `reasoning_mode` pour pouvoir indiquer le comportement du nœud à l'initialisation des simulations.

- `reasoning_mode == 3` pour le premier cas,
- `reasoning_mode == 4` pour le second cas.

Comme nous l'avons expliqué dans la section 4.2.1, le principe de l'approche repose sur le fait de retarder la prise de décision pour récolter des informations supplémentaires à travers le comportement des voisins et ensuite se fier à l'avis de la majorité. La question qui se pose alors : pendant combien de temps un nœud devrait attendre avant de déclencher cette procédure pour qu'il soit sûr d'avoir la majorité des messages des voisins. Nous proposons de fixer cette valeur dans ce cas à 6 milli-secondes.

```
DECISION_TIMEOUT = 6 ms
```

Ce temps correspond à la durée moyenne obtenue en calculant la différence entre la réception pour la première fois et la dernière fois d'une demande de route bien déterminée.

Nous présentons la mise en place de chaque cas de figure.

4.2.2.1 Cas 1 : décider puis traiter

À la réception d'une demande de route pour la première fois, le nœud ne la traite pas et associe à cette demande un `timer` initialisé à `DECISION_TIMEOUT`. Il enregistre ensuite toutes les demandes de route similaires (identifiées par le même couple [`#ID`, `@SRC`]) au niveau de sa table d'historique étendue (THE).

Une procédure de décision est invoquée à la fin du décompte du `timer`. Elle se charge de classer les demandes de route ayant les mêmes identifiants par rapport au numéro de séquence de la source (#SNS) et décide selon le nombre de groupes obtenus :

- S'il n'y a qu'un seul groupe cela veut dire que toutes les demandes de route reçues portent le même numéro de séquence source et ceci prouve qu'il n'y a pas d'actions malhonnêtes. Le nœud continue alors le traitement de la première demande de route reçue tel que présenté dans la spécification.
- S'il y a plus d'un groupe, ceci prouve l'existence d'action malhonnête. Le nœud se tourne alors vers l'avis de la majorité pour décider. Il considère que le groupe ayant le plus grand nombre de requêtes reçus comme étant composé de nœuds honnêtes et ainsi il peut accuser les nœuds envoyant les requêtes appartenant à l'autre groupe de malhonnêteté. Le nœud peut alors continuer le traitement de la première demande de route reçue faisant partie du groupe de nœuds considérés honnêtes.

  Cependant dans certains cas, nous obtenons un nombre de requêtes équivalent dans les groupes, ce qui ne nous permet pas de prendre de décision. Dans ce cas, le nœud choisi une requête au hasard qu'il traite en suivant la spécification du protocole.

Ce traitement nécessite la modification de la procédure `rreq_process` qui doit être divisée en deux parties :

- La première partie sert au stockage des informations et la mise à jour des chemins dans la table de routage ce qui correspond aux lignes 1 à 14 de l'algorithme C.1 page 122. Ceci permet à chaque nœud de récupérer les demandes de route similaires pendant un temps égal à 6 ms.
- La seconde partie sert à la retransmission de la demande de route ou de la réponse de route (ligne 15 à 27 de l'algorithme C.1 page 122). Cette partie fait l'objet d'une nouvelle procédure dénotée `rreq_trait`.

Nous présentons sous forme de pseudo-code la mise en place de la procédure de décision

invoquée à la fin du temps `DECISION_TIMEOUT` .

```
Procédure DecisionTimeout(RREQ rreq)
Début
    L = Rechercher les occurrences de la rreq dans la THE
    Classer les messages de (L) en groupes par rapport à #SNS
    Si (Nombre de groupes == 1) Alors
        first_rreq = Chercher la première requête reçue
        rreq_trait(first_rreq)
    Si (Nombre de groupes == 2) Alors
        Si (Nombre d'éléments équivalent dans chaque groupe) Alors
            Pas de décision
            first_rreq = Chercher une requête au hasard
            rreq_trait(first_rreq)
        Sinon
            Si (Le groupe 1 contient plus d'éléments que le groupe 2) Alors
                Accuser les nœuds initiant les rreq du groupe 2
                first_rreq = Chercher la première requête reçue du groupe 1
                rreq_trait(first_rreq)
            Sinon
                Accuser les nœuds initiant les rreq du groupe 1
                first_rreq = Chercher la première requête reçue du groupe 2
                rreq_trait(first_rreq)
```

4.2.2.2 Cas 2 : traiter puis décider

La différence par rapport au premier cas réside dans le fait de traiter normalement la première demande de route dès que reçue en associant un `timer` initialisé à `DECISION_TIMEOUT`. Lors de la réception de demandes de route similaires (identifiées par le même couple [`#ID`, `@SRC`]), le nœud les enregistre au niveau de sa table d'historique étendue (THE).

À la fin du décompte du `timer`, une procédure de décision est invoquée : elle parcourt la table d'historique étendue à la recherche de toutes les demandes de route reçues des voisins ayant les mêmes identifiants et les classent par rapport au numéro de séquence de la source (#SNS). Ainsi, selon le nombre de groupes obtenus, un nœud pourra décider :

- S'il n'y a qu'un seul groupe cela veut dire que toutes les demandes de route reçues portent le même numéro de séquence source et ceci prouve qu'il n'y a pas d'actions malhonnêtes.
- S'il y a plus d'un groupe, ceci prouve l'existence d'action malhonnête. Le nœud se tourne alors vers l'avis de la majorité pour décider. Il considère que le groupe ayant le plus grand nombre de requêtes reçues comme étant composé de nœuds honnêtes et ainsi il peut accuser les nœuds envoyant les requêtes appartenant à l'autre groupe de malhonnêteté. Cependant dans certains cas, nous obtenons un nombre de requêtes équivalent dans les groupes, ce qui

ne nous permet pas de prendre de décision.

Nous présentons sous forme de pseudo-code le traitement exécuté lorsque le temps d'attente est écoulé pour prendre une décision sur le comportement des nœuds voisins.

```
Procédure DecisionTimeout(RREQ rreq)
Début
    L = Rechercher les occurrences de la rreq dans la THE
    Classer les messages de (L) en groupes par rapport à #SNS
    Si (Nombre de groupes == 2) Alors
        Si (Nombre d'éléments équivalent dans chaque groupe) Alors
            Pas de décision
        Sinon
            Si (Le groupe 1 contient plus d'éléments que le groupe 2) Alors
                Accuser les nœuds initiant les rreq du groupe 2
            Sinon
                Accuser les nœuds initiant les rreq du groupe 1
```

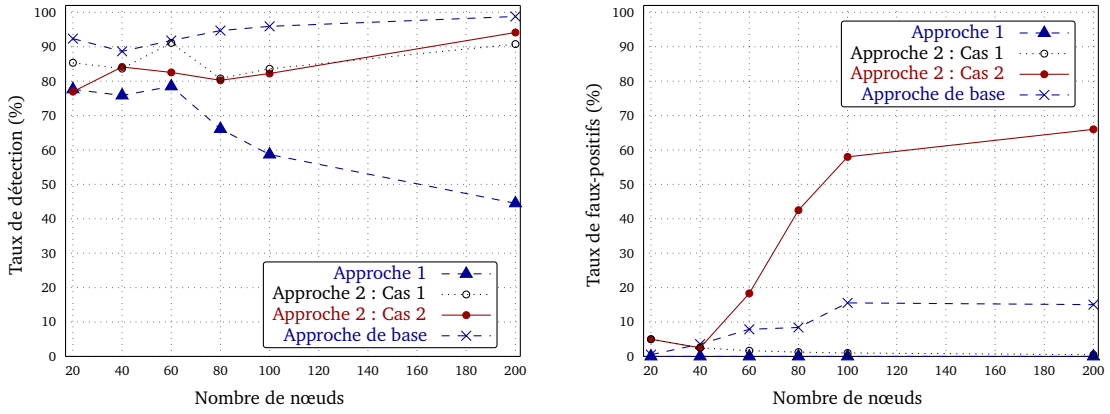
D'autres modifications ont été introduites pour ajouter le nouveau raisonnement. Ils portent essentiellement sur les contrôles permettant l'exécution des règles. Il est à noter que nous combinons la gestion d'une liste noire permanente et d'une liste noire temporaire permettant de punir les nœuds selon que l'accusation est sûre ou non.

4.2.3 Résultats expérimentaux

Comme c'est le cas pour l'approche 1, nous considérons les mêmes paramètres présentés dans la section 3.5 pour l'exécution des simulations. Nous utilisons le taux de détection des actions malhonnêtes ainsi que le taux de faux-positifs pour évaluer la performance du système de détection. Nous varions ainsi :

– **La densité du réseau** qui correspond à la variation du nombre de nœuds de 20 à 200 tout en gardant constant le trafic généré entre les nœuds (7 paires d'émetteurs/récepteurs). Le nombre d'attaquants est fixé à un.

  La figure 4.5 représente les résultats obtenus (taux de détection et taux de faux-positifs) pour l'attaque par modification du numéro de séquence de la source dans une demande de route pour (i) l'approche de base, (ii) l'approche 1, et (iii) pour l'approche 1 associée au complément de détection décrit dans cette section (noté approche 2) avec ses deux cas de figure. Comme nous pouvons le constater dans la figure 4.5a, le taux de détection de l'approche 2 (courbes marquées par des cercles pleins et cercles vides) est plus important que celui de l'approche 1 (courbe marquée par des triangles). Ce résultat est prévisible puisque le complément que nous avons mis en place s'ajoute à l'approche 1 pour traiter les cas où une décision ne peut être prise ce qui nous permet de prévoir un taux de détection au moins équivalent à celui de l'attaque 1. Cependant, le taux de détection reste inférieur à celui de l'approche de base.



a. Variation du taux de détection

b. Variation du taux de faux-positifs

FIGURE 4.5 – Impact de la variation de la densité du réseau en utilisant le complément de détection

L'amélioration constatée que ce complément propose pour la détection d'actions malhonnêtes est associée à une baisse du nombre de faux-positifs dans le premier cas par rapport à l'approche de base alors qu'il engendre un nombre important de faux-positifs dans le second cas. La figure 4.5b montre la variation de ces fausses accusations en variant la densité du réseau.

Le nombre important des fausses accusations obtenu dans le second cas de figure est dû aux cas où la procédure de décision mise en place ne prend aucune décision à cause de l'égalité des messages dans chaque groupe ce qui propage l'attaque et fait que certains nœuds reçoivent un nombre plus grand de messages erronés faussant ainsi la décision. Ceci associé à une densité croissante accentue le nombre de nœuds accusés à tort.

La figure 4.6 présente l'évolution du taux de paquets reçus avec succès (PDR) en variant la densité du réseau pour les deux cas de figure de l'approche 2. Les résultats obtenus montrent que le premier cas de figure (courbe avec des cercles pleins) dégrade les performances du protocole de base (courbe avec des triangles pleins). Cependant, le second cas de figure présente une amélioration du même taux par rapport au protocole de base pour des faibles densités.

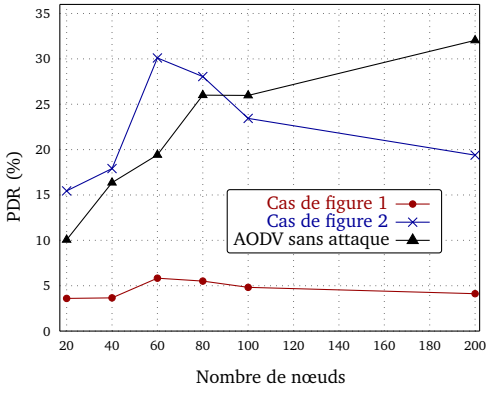
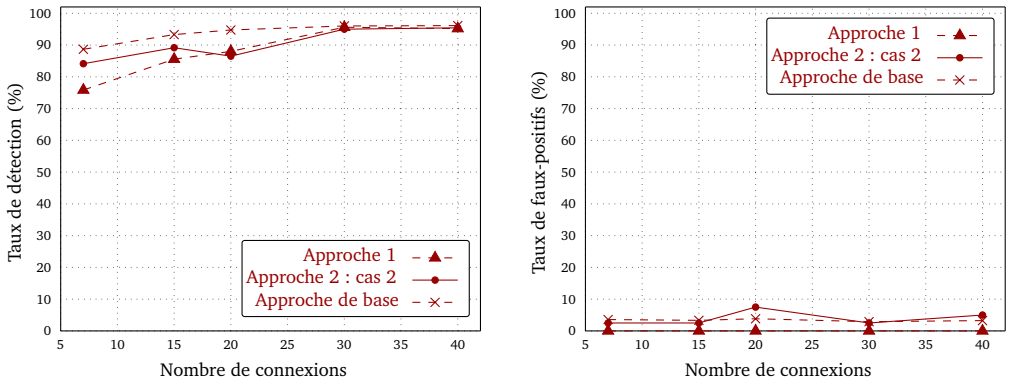


FIGURE 4.6 – Influence de la variation de la densité du réseau sur le PDR

– **La densité du trafic** : Nous varions le nombre de connexions de 7 à 40 dans un réseau de 40 nœuds (figure 4.7) et de 7 à 200 dans un réseau de 100 nœuds (figure 4.8). Le nombre d'attaquants est le même dans les deux cas.

  La variation de la densité du trafic a une faible influence sur le taux de détection. Nous remarquons une amélioration de ce taux avec un trafic faible et qui s'approche des taux obtenus avec l'approche 1 pour les densités les plus fortes.

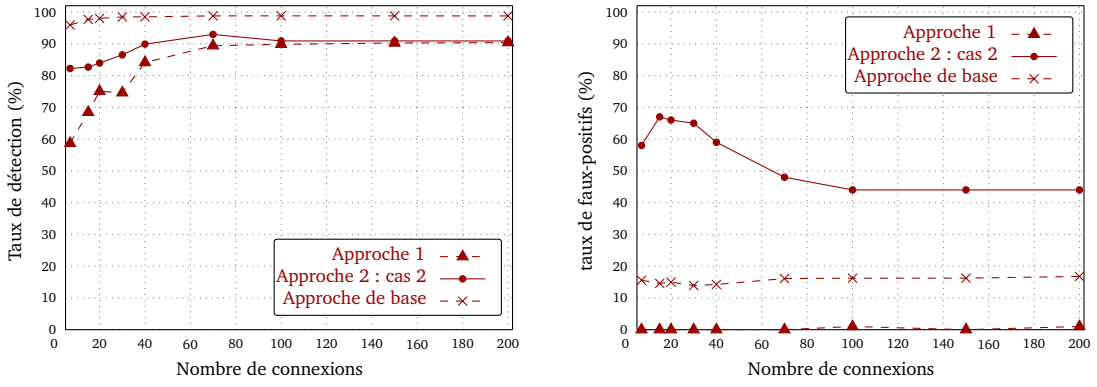
  Comme c'est le cas lors de la variation de la densité du réseau, la variation du trafic engendre des faux-positifs. Ces cas sont plus importants pour des réseaux plus denses comme le montre les figures 4.7b pour un réseau de 40 nœuds et 4.8b pour un réseau de 100 nœuds.



a. Variation du taux de détection

b. Variation du taux de faux-positifs

FIGURE 4.7 – Impact de la variation de la densité du trafic dans un réseau de 40 nœuds en utilisant le complément de détection



a. Variation du taux de détection

b. Variation du taux de faux-positifs

FIGURE 4.8 – Impact de la variation de la densité du trafic dans un réseau de 100 nœuds en utilisant le complément de détection

Comme nous pouvons le constater lors de ces expérimentations, le complément de détection utilisant le premier cas de figure améliore la détection des nœuds malhonnêtes. Toutefois, il dégrade les performances du protocole constatée en évaluant le taux de paquets reçus avec succès.

Le complément de détection utilisant le second cas de figure propose aussi une amélioration des taux de détection sauf qu'il génère un nombre important de faux-positifs. Nous concluons que cette approche de détection n'est pas appropriée. Elle le sera peut être en mettant en place un échange d'information entre les nœuds à propos des accusations permettant ainsi aux nœuds d'avoir plus d'informations qu'ils peuvent utiliser pour prendre la bonne décision.

Notons que le nombre de groupes est limité à deux dans les cas traités. Ceci s'explique par le fait qu'il n'y a qu'un seul attaquant dans le réseau. Celui-ci n'intervient qu'une seule fois sur une demande de route en modifiant le `#SNS`. Toutefois, en augmentant le nombre d'attaquants, nous obtenons plus de groupes. Cependant, le principe du complément de détection décrit dans cette section reste le même.

Dans ce qui suit, nous discutons les mesures prises à l'encontre des nœuds malhonnêtes en montrant l'impact de ces mesures sur les performances du système de détection.

4.3 Mesures prises à l'encontre des nœuds malhonnêtes

4.3.1 Types de mesures

Dans cette section, nous allons débattre de la question suivante : faut-il bannir temporairement ou définitivement un nœud soupçonné de malhonnêteté ?

4.3.1.1 Exclusion temporaire

Ce choix a l'avantage de bannir les nœuds accusés par erreur pendant une courte durée : un intervalle de temps au delà duquel le nœud puni peut à nouveau interagir avec ses voisins comme auparavant. Malheureusement, ceci est vrai pour les nœuds malhonnêtes qui en profitent pour perturber encore une fois le déroulement des opérations.

Nous avons choisi de bannir temporairement les nœuds soupçonnés de malhonnêteté parce qu'il y a des cas de faux positifs ce qui permet au nœuds accusés à tort d'interagir à nouveau avec les autres. L'intervalle de temps durant lequel les nœuds sont bannis doit être au moins égal à `BLACKLIST_TIMEOUT`. Celui-ci correspond au temps nécessaire à l'établissement d'un chemin ce qui évite de prendre en considération les messages du nœud malhonnête lors de la formation du chemin. Ce paramètre prend la valeur suivante :

$$\texttt{BLACKLIST\_TIMEOUT} = 2 \times \texttt{NET\_TRAVERSAL\_TIME}$$

$$\text{avec } \texttt{NET\_TRAVERSAL\_TIME} = 2 \times \texttt{NODE\_TRAVERSAL\_TIME} \times \texttt{NET\_DIAMETER}$$

Nous excluons ainsi le nœud malhonnête pendant au moins le temps qu'une nouvelle route qui ne passe pas par lui soit créée. Il est à noter que le calcul de ce temps dépend du diamètre du réseau (`NET_DIAMETER`) qui peut être difficile à évaluer dans certains cas.

4.3.1.2 Exclusion définitive

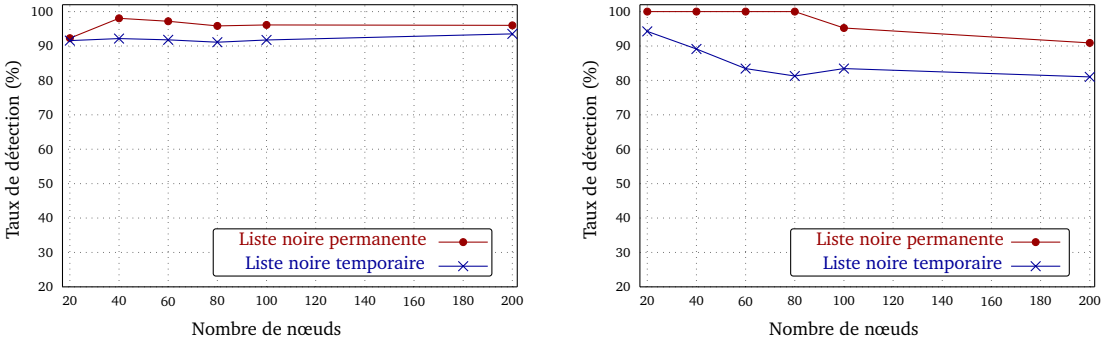
Dans ce cas, si un nœud est détecté comme malveillant, ses messages ne sont plus pris en compte et sont automatiquement rejetés à n'importe quel moment ce qui évite d'avoir à surveiller ce nœud. Ce type d'exclusion doit être mis en place seulement lorsque le nœud est sûr de son accusation.

D'un autre côté, cette action semble sévère dans le cas où un nœud est détecté par erreur. Il nous semble dès lors plus judicieux de parler d'exclusion de longue durée. Ainsi, un nœud est banni du réseau assez longtemps sans participer aux opérations du réseau. Toutefois, il reste à déterminer cette durée d'exclusion : il faut qu'elle soit choisie de telle sorte à ce que le taux de détection obtenu soit au plus prêt de celui obtenu en utilisant les listes noires définitives.

Dans ce qui suit, nous proposons de présenter les résultats obtenus en utilisant une liste noire permanente et une liste noire temporaire.

4.3.2 Comparaison de l'impact de chaque mesure

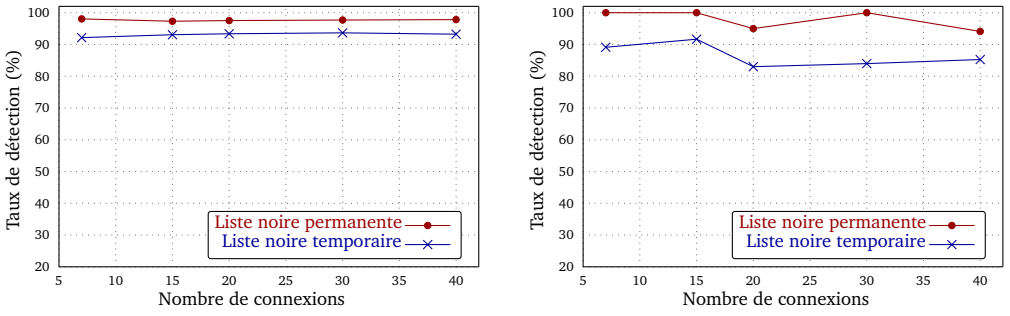
Les figures 4.9 et 4.10 montrent la différence obtenue dans les taux de détection pour les attaques par rejeu (sur RREQ et RREP) en variant la densité du réseau et la densité du trafic.



a. Attaque par rejeu de RREQ

b. Attaque par rejeu de RREP

FIGURE 4.9 – Impact de la variation de la densité du réseau



a. Attaque par rejeu de RREQ

b. Attaque par rejeu de RREP

FIGURE 4.10 – Impact de la variation de la densité du trafic dans un réseau de 40 nœuds

Les résultats obtenus montrent que l'utilisation d'une liste noire permanente améliore les taux de détection. Ceci est dû essentiellement au fait que les nœuds n'oublient pas les nœuds malhonnêtes comme c'est le cas des nœuds utilisant les listes noires temporaires et réussissent ainsi à bloquer leurs attaques sans avoir à retrouver une nouvelle preuve d'accusation.

Il est à noter que nous nous permettons la mise en place d'une liste noire permanente parce que notre système de détection génère peu de faux positifs dans ces cas.

La combinaison des deux types d'exclusion des nœuds est une alternative intéressante à mettre en place. Dans ce cas, le bannissement des nœuds dépendra du degré de certitude du nœud accusateur sur la culpabilité de l'émetteur du message : il opte pour un bannissement de longue durée si l'action malhonnête est prouvée et pour un bannissement temporaire en cas de doute.

4.4 Résumé

Nous avons établi que l'origine majeure des problèmes de faux positifs tient à la mauvaise interprétation du critère d'incohérence 2.4. Nous avons proposé dans ce chapitre deux solutions pour limiter ce problème. Elle se base sur le fait de ne prendre une décision que si le nœud est sûr de l'action malhonnête du voisin. Ainsi, un nœud n'est capable de prendre une décision sûre que s'il est la source du message ou s'il est voisin de la source.

Après avoir présenté les modifications apportées au critère d'incohérence 2.4, nous avons passé en revue sa mise en place. Nous avons expliqué comment intégrer cette solution à l'implémentation actuelle de telle sorte à pouvoir l'exécuter en parallèle avec l'ancienne version.

Après exécution des simulations, nous avons présenté les résultats obtenus. Comme c'est le cas dans le chapitre 3, nous avons commencé cette analyse par l'étude des performances du système de détection. Nous avons utilisé pour ceci le paramètre taux de détection associé au taux de faux positifs. Ainsi, nous avons varié la densité du réseau dans un premier temps et la densité du trafic dans un second temps. Nous avons testé ensuite l'efficacité de la nouvelle implémentation face à l'augmentation du nombre de nœuds malhonnêtes. Les résultats obtenus ont montré que cette solution limite les fausses accusations tout en assurant un taux de détection élevé. Cependant, ces taux restent moins importants que ceux de l'approche de base. Notons qu'aux vues des expérimentations, l'intérêt de cette solution s'exprime principalement dans des réseaux à faible densité. Nous avons terminé l'analyse par l'étude des performances du protocole et avons montré que cette solution apporte encore une amélioration à l'ancienne version, qui offrait déjà de meilleures performances que AODV de base.

Nous nous sommes intéressés ensuite à améliorer la détection des nœuds malhonnêtes pour la solution mettant en place le critère 2.4' puisque ce taux est inférieur à celui de la solution de base. Ce complément s'appuie sur le fait de se fier à l'avis de la majorité lors de la décision. Nous distinguons deux cas de figure : dans le premier le nœud stocke, décide ensuite traite la bonne demande de route et dans le second traite normalement la première RREQ, stocke ensuite les demandes similaires et décide enfin de l'honnêteté des nœuds. Les résultats obtenus après simulations ont montré que malgré l'amélioration du taux de détection (qui se rapproche des taux obtenus pour l'approche de base), nous avons décelé une dégradation importante des performances du protocole dans le premier cas et un nombre important de faux positifs dans le second cas ce qui nous a permis de conclure que cette approche n'est pas appropriée.

Nous avons enfin discuté des mesures qu'un nœud peut entreprendre à l'encontre de nœuds malhonnêtes. Nous avons distingué ainsi entre deux types de mesures :

– Des mesures *temporaires* en excluant les nœuds malhonnêtes pendant un intervalle de temps qui est fixé à au moins `BLACKLIST_TIMEOUT`.

– Des mesures *radicales* en bannissant définitivement les nœuds dont l'action malhonnête est décelée.

Les expérimentations effectuées pour comparer entre ces mesures ont montré qu'en bannissant définitivement les nœuds malhonnêtes, les performances de détection sont améliorées par rapport à une exclusion temporaire. Cependant, les mesures radicales peuvent être injuste lorsqu'un nœud est détecté par erreur ce qui nous a amené à proposer de varier le temps d'exclusion selon les cas : un nœud est banni pendant une longue durée si le nœud est sûr de l'action malhonnête et pendant une courte durée sinon.

CONCLUSION

Bilan

Dans cette thèse, nous nous sommes intéressés à la sécurité des protocoles de routage ad hoc. L'objectif était de proposer des mécanismes de détection d'actions malhonnêtes pour consolider les protocoles de routage ad hoc et prévenir les attaques. Plus précisément, nous avons basé notre étude sur le protocole AODV en nous appuyant sur la confiance que les nœuds partagent afin de mettre en place des mécanismes permettant de suivre l'exécution du protocole et de déceler d'éventuelles incohérences.

Dans le contexte de réseaux ad hoc, la notion de confiance est plus sollicitée au vu des caractéristiques particulières de ce type de réseaux (mobilité, distribué, auto-organisé, ad hoc) où les nœuds communiquent entre eux sans connaissance préalable de l'identité ou même de la topologie. Nous avons opté pour la mise en place au niveau de chaque nœud d'un raisonnement sur le comportement des voisins en utilisant le recoupement d'informations fraîchement reçues avec celles reçues précédemment et ainsi détecter les incohérences. Ce travail a suivi les étapes suivantes :

Analyse de la confiance dans AODV. Cette analyse a utilisé un langage formel qui se base sur celui proposé par [YKB93] pour la modélisation des règles de confiance dans AODV. Nous nous sommes focalisé sur le comportement interne d'un nœud à chaque étape du processus de découverte et de maintient de route pour dégager les règles qu'un nœud suit lors du traitement d'un message de routage.

Les règles ainsi obtenues annonçaient la relation de confiance entre un nœud et le voisin duquel le message en cours de traitement est reçu puisque ce processus n'aboutit que si les nœuds coopèrent et croient en l'exactitude des informations fournies. Elles intégraient aussi le traitement adéquat à chaque situation.

Détection des nœuds malhonnêtes. Le principe de détection que nous avons proposé repose sur le contrôle du comportement du voisinage en recoupant les informations fraîchement reçues avec celles précédemment observées, ce qui permet de déceler d'éventuelles incohérences. Toutefois, la connaissance acquise par les nœuds lors du processus de routage AODV de base ne suffisait pas à vérifier certains comportements malhonnêtes, ce qui nous a amené à créer une structure stockant un historique étendu des messages reçus. Cette nouvelle base de connaissance a servi à la vérification de la cohérence des informations reçues.

Nous avons ainsi formalisé des critères notés critères d'incohérences, qui correspondaient à des contrôles à intégrer au niveau de chaque nœud. Ces critères permettaient de déceler des actions malhonnêtes élémentaires (rejeu, fabrication et modification de messages). Ces contrôles une fois mis en place ajoutent une ligne de défense permettant la détection en temps réel d'attaques contre les messages de routage.

Expérimentations. L'évaluation de notre système de détection était une étape importante pour valider le raisonnement suivi et montrer son efficacité. Durant cette étape, nous avons opté pour l'utilisation du simulateur NS-2 [IH08] et l'implémentation AODV-UU [Nor07].

Pour pouvoir tester l'extension, il fallait mettre en place un comportement malhonnête. Nous avons ainsi créé un attaquant pouvant utiliser les vulnérabilités du protocole pour aboutir à ses fins. Son action malhonnête reposait essentiellement sur des actions élémentaires et sur la composition de ceux-ci.

L'évaluation a porté ensuite sur trois axes :

- Évaluation des performances du système de détection où nous avons utilisé la métrique du taux de détection qui établissait la proportion de nœuds dans le voisinage d'un nœud malhonnête qui pouvaient découvrir l'action malveillante. Nous avons ainsi varié la densité du réseau et la densité du trafic tout en observant le comportement du système. Après analyse des résultats, nous avons pu affirmer l'efficacité de l'extension que nous avons proposé : nous avons obtenu des taux de détection variant entre 70% et 98% et de faibles taux de faux positifs variant entre 0 et 15%.
- Évaluation des performances du protocole : nous avons évalué le taux de paquets reçus avec succès, le délais de bout-en-bout et la charge de routage pour vérifier les performances du protocole. Les résultats ont montré que l'exécution du protocole étendu (même en cas d'attaques) avait des performances au moins semblables voire meilleures dans certains cas que le protocole de base.
- Évaluation de la limite du système de détection : nous avons testé les performances de détection dans un environnement fortement corrompu. Les résultats ont démontré une dégradation des taux de détection et une augmentation des cas de faux positifs. Nous avons conclu qu'au delà de 25% de nœuds malveillants dans le réseau le système de détection n'etait plus fiable puisque le taux de détection était inférieur à 60%.

Cette analyse nous a permis de montrer l'efficacité du système de détection que nous proposions ainsi que ses limites. Cependant, les cas de faux positifs qui sont apparus nous ont interpelé. C'est pourquoi nous nous sommes intéressé à l'étude de leurs causes afin de les supprimer.

Étude des faux-positifs. Certains nœuds prenaient de mauvaises décisions (précisément lors de l'exécution du critère d'incohérence 2.4) en accusant des nœuds innocents. Pour remédier à ce problème, nous avons réduit, pour le critère d'incohérence concerné, les cas où un nœud peut prendre la décision d'accuser son voisin. Nous avons ajouté l'obligation d'avoir une preuve irréfutable de l'action malhonnête. Nous avons ainsi reformulé ce critère d'incohérence, puis nous l'avons implémenté pour pouvoir effectuer les simulations. Les résultats confirmaient les attentes puisqu'il n'y avait plus de faux positifs. Cependant, nous avons observé une dégradation des taux de détection allant jusqu'à 20% en moins. Ceci nous a amené à proposer un complément de détection : la technique reposait sur le fait de retarder la décision jusqu'à rassembler tous les messages du voisinage et se fier à l'avis de la majorité pour formuler une décision. Après implémentation, les résultats des expérimentations ont montré que malgré la validité du raisonnement qui a permis d'augmenter le taux de détection, cette solution est à éviter puisqu'elle générait un nombre conséquent de faux positifs.

Nous avons aussi étudié les mesures qu'un nœud peut prendre à l'encontre des nœuds malhonnêtes. Nous avons ainsi comparé une exclusion temporaire à une exclusion définitive d'un nœud. Les résultats obtenus en bannissant définitivement les nœuds malhonnêtes sont légèrement supérieurs à ceux obtenus en bannissant temporairement ces nœuds. Toutefois, en jouant sur le temps de bannissement (allonger/écourter) d'un nœud, on peut trouver un compromis entre les deux.

Perspectives

Le travail que nous avons accompli ouvre des perspectives laissant envisager un certain nombre d'études complémentaires dans les directions suivantes :

Tester le système de détection dans un environnement réel. Il serait intéressant de voir le comportement du système dans un environnement réel. Ceci vient compléter les résultats obtenus par simulations. Ce travail ne devrait pas poser de problèmes puisque l'implémentation `AODV-UU` choisie du protocole de routage AODV s'intègre dans NS-2 et dans le noyau linux.

Améliorer le système de détection mis en place. La mise en place d'une politique de gestion des mesures prises à l'encontre des nœuds malhonnêtes peut contribuer dans l'amélioration des performances de détection. Ainsi, nous envisageons la recherche d'une valeur *optimale* pour l'intervalle de temps durant lequel les nœuds sont bannis de telle façon à se rapprocher le plus des taux de détection avec un bannissement définitif sans pour autant tomber dans ses inconvénients ou aussi la combinaison de bannissement à intervalle de temps variable (longue/courte durée).

Nous pouvons aussi envisager d'étendre la connaissance de chaque nœud en stockant localement une vision propre de la table de routage et de la table d'historique des voisins. Il faudra alors démontrer l'utilité de telles informations et justifier leurs ajout par des règles permettant la détection de nouvelles actions malhonnêtes.

Échanger les informations. L'échange des données concernant les incohérences décelées tout en fournissant les preuves d'accusation peut aider les nœuds à mieux se protéger. Ainsi, un protocole d'échange de ce type de données doit être mis en place pour partager les données entre les nœuds.

Généraliser l'approche. Il nous parait intéressant de porter le raisonnement vers d'autres protocoles réactifs. Nous proposons ensuite la création d'un modèle générique pour les protocoles réactifs auquel il faut associer un ensemble de règles de confiance. La difficulté de ce travail réside dans le fait de créer le modèle générique pour les protocoles de routage réactifs. Il suffit ensuite de trouver un mappage entre le modèle générique et le protocole lui même. Ce qui permettra d'adapter le raisonnement à n'importe quel protocole réactif.

Nous pensons aussi que, dans le contexte des réseaux ad hoc, les mécanismes déployés ne doivent pas se limiter à la couche réseau, mais s'étendre au niveau physique (accès au médium) et applicatif (services fournis). Ainsi, il serait intéressant de mettre en place des mécanismes permettant de suivre l'exécution des protocoles de routage (via une analyse protocolaire) en vue de comparer les messages échangés entre les nœuds du réseau ad hoc.

ANNEXE A

RAPPEL DES NOTIONS CRYPTOGRAPHIQUES

La cryptographie est l'ensemble des méthodes permettant de sécuriser le transfert des données de telle sorte que seulement les nœuds concernés puissent retrouver les données initiales. Un message est transformé pour le rendre incompréhensible pour toute personne non autorisée et empêche toute modification. Le destinataire peut récupérer les données en claire et s'assurer que celles-ci n'ont pas été altérées. Les transformations en question sont basées sur des fonctions mathématiques appelées algorithmes cryptographiques.

La cryptographie permet d'assurer les propriétés de sécurité suivantes :

- La confidentialité : le message est incompréhensible à toute personne non autorisée ;
- L'intégrité : le message protégé n'est pas modifié par une entité non autorisé ;
- L'authenticité : le message provient de l'émetteur revendiqué et l'information reçue est intègre.

Les algorithmes cryptographiques de chiffrement sont constitués de deux opérations symétriques :

- Le *chiffrement* : opération permettant de transformer un message en clair en un message chiffré (incompréhensible) ;
- Le *déchiffrement* : opération symétrique à celle de chiffrement permettant de retrouver le message en clair à partir du chiffré.

Puisque les algorithmes de chiffrement/déchiffrement sont publics (donc *a priori* connus de tous), le secret repose exclusivement sur le paramètre *clé* [Ker83]. Ainsi, les entités capables de déchiffrer les messages connaissent une information supplémentaire capitale dans ce processus

Nous distinguons deux types d'algorithmes cryptographiques réversibles : ceux utilisant la même clé (symétrique) et ceux utilisant deux clés différentes (asymétrique).

A.1 Cryptographie symétrique

Les mécanismes de chiffrement symétriques à clé secrète, permettent à des entités (généralement deux) d'effectuer des opérations de chiffrement/déchiffrement en utilisant une seule clé. En ce sens, toute personne qui peut chiffrer des messages et donc qui connait la clé secrète, peut également déchiffrer. Ce mode de chiffrement se comporte comme une boite fermé avec une serrure où les entités désirant communiquer doivent posséder chacun une clé leur permettant d'ouvrir et fermer la boite.

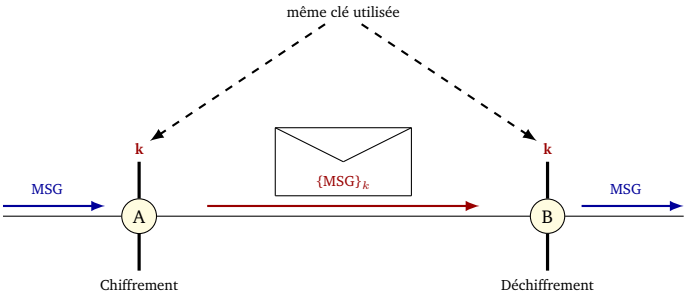


FIGURE A.1 – Chiffrement symétrique

Dans la figure A.1, le nœud A désire envoyer un message à B. Il chiffre donc son message avec la clé k. En recevant le message, B déchiffre en utilisant la même clé k. Il est important de noter qu'il y a une phase initiale ou les deux nœuds se mettent d'accord sur la clé secrète qui sera utilisée.

Ce mode de protection de la confidentialité des données présente l'avantage d'être peu gourmand en terme de puissance de calcul. Cependant, son principal inconvénient réside dans la distribution de la clé secrète avant de commencer la communication. Cette phase d'échange de clés est d'autant plus importante lorsque les entités désirant communiquer de manière sécurisée en point à point sont de plus en plus nombreuses.

Parmi les algorithmes de chiffrement symétrique les plus connus, nous citons AES [DR02] (*Advanced Encryption Standard*) ou encore DES [Kah03] (Data Encryption Standard).

A.2 Cryptographie asymétrique

On peut assimiler les mécanismes à clés publique à un coffre fort dont une seule personne possède la clé. Ce coffre est laissé ouvert à la disposition de toute personne désirant lui envoyer un message. Une fois fermé, seule le propriétaire du coffre pourra l'ouvrir.

En pratique, il s'agit d'une fonction de chiffrement à deux clés : une clé publique qui est fourni à tous ceux désirant envoyer un message confidentiel à une entité et une clé privée qui sert à déchiffrer les messages qui lui sont destinés. Ces deux clés sont liées de telle sorte que seule la clé privée peut déchiffrer un message chiffré par la clé publique correspondante.

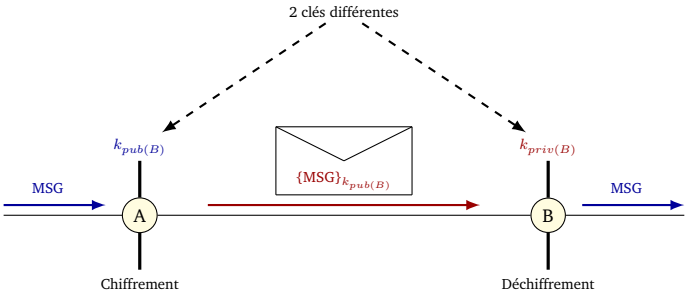


FIGURE A.2 – Chiffrement asymétrique

Dans la figure A.2, le nœud A désire transmettre le message MSG à B. Il récupère alors la clé publique de la destination $k_{pub(B)}$ et chiffre le message en utilisant cette clé ($\{MSG\}_{k_{pub(B)}}$). Le message chiffré est envoyé sur le canal de communication. La destination B déchiffre le message reçu en utilisant sa clé secrète $k_{priv(B)}$ pour obtenir le message en clair envoyé par A.

Un des algorithmes à clé publique les plus populaires est RSA [RSA78] du nom de ses concepteurs Rivest, Shamir et Adleman. Cet algorithme est basé sur le problème de factorisation d'entiers.

La cryptographie asymétrique nécessite beaucoup plus de puissance de calcul par rapport à la cryptographie symétrique. C'est pourquoi, elle est principalement utilisée dans la phase d'échange de clé de session symétrique entre les entités, clé qui est utilisée par la suite pour chiffrer les communications.

En terme d'authenticité, la cryptographie asymétrique est la seule à garantir véritablement l'authentification de la source puisque la cryptographie symétrique ne distingue pas la source de la destination. Nous décrivons dans la section A.5 le principe de la signature numérique largement utilisée comme schéma d'authentification.

A.3 Fonctions de hachage cryptographiques

Pour garantir l'intégrité des messages, une fonction de hachage cryptographique peut être utilisée. C'est une fonction non réversible (*one-way*) produisant un condensé (appelé aussi empreinte ou haché) ayant les propriétés suivantes :

- Unique et de taille fixe ;
- Il est impossible (techniquement parlant et dans un temps raisonnable) de retrouver le message d'origine à partir du condensé ;
- Sachant un message donné et son empreinte en utilisant une fonction de hachage, il est très difficile de générer un autre message qui donne la même empreinte (*weak collision resistance*) ;
- Il est impossible de trouver deux messages produisant le même condensé (*strong collision resistance*).

Le temps négligeable pour le calcul du condensé constitue un avantage pour l'utilisation des fonctions de hachages cryptographiques. Ces fonctions sont utilisées entre autres pour la signature numérique (le condensé du message est calculé et lui seul est signé) et aussi pour des mécanismes

d'authentification par mot de passe sans stockage de ce dernier.

La figure A.3 montre l'utilisation de la fonction de hachage pour préserver l'intégrité des messages échangés entre deux entités (A et B).

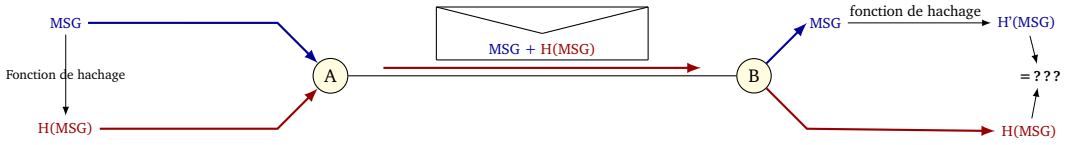


FIGURE A.3 – Utilisation des fonctions de hachage cryptographique

A.4 Message Authentication Code

L'algorithme MAC (*Message Authentication Code*) est un cas particulier de fonction de hachage. Comme une fonction de hachage, l'algorithme MAC produit un condensé de taille fixe (appelé code d'authentification de message - MAC). Mais à la différence de celui-ci, il utilise en plus une clé secrète.

Le MAC peut être utilisé pour vérifier simultanément l'intégrité de données comme tout autre fonction de hachage et l'authenticité de la source des données.

L'émetteur calcule le MAC en utilisant la fonction de hachage qui prend en paramètre la symétrique k. Ensuite, le message et le MAC sont transmis au destinataire. Celui-ci calcule le MAC du message reçu en appliquant la même fonction de hachage avec la clé symétrique k et compare le résultat obtenu avec le MAC reçu. S'il y a égalité, le message n'a pas été altéré et il provient bien du nœud supposé (voir figure A.4).

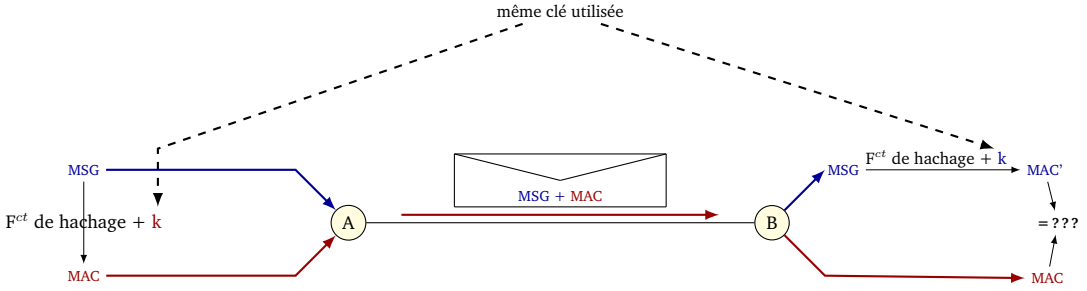


FIGURE A.4 – Utilisation des fonctions de hachage à clé symétrique

Dans la pratique, une méthode de calcul du MAC à base de fonction de hachage plus élaborée et plus sûre est utilisée : il s'agit du HMAC (*keyed-Hash Message Authentication Code* [KBC97]) qui a fait l'objet de la RFC 2104.

A.5 Signature numérique

La signature numérique est un mécanisme assurant l'intégrité d'un message ainsi que l'authentification de la source de celui-ci. Ceci est possible grâce à la cryptographie asymétrique.

Lorsqu'une source désire transmettre des données à une destination, elle applique la fonction de hachage sur le message à envoyer et le condensé obtenu est chiffré avec sa clé privée. Le résultat obtenu constitue la signature et est envoyé avec le message (voir figure A.5). La destination utilise la clé publique de la source pour déchiffrer le condensé et le compare au condensé obtenu en appliquant la fonction de hachage sur le message reçu. Ainsi, la destination garantit que le message vient de la source et non pas d'un nœud usurpant son identité et garantit aussi que le message n'a pas été altéré lors du transfert.

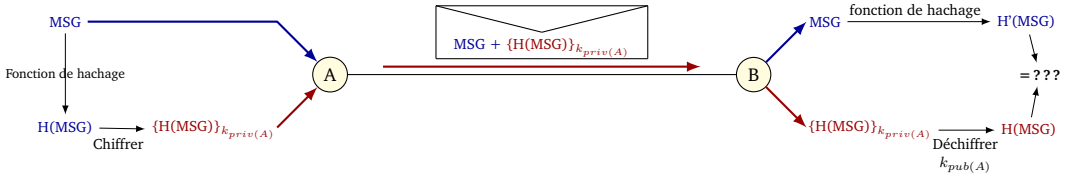


FIGURE A.5 – Signature numérique

A.6 Certificat électronique

Un certificat électronique est une carte d'identité numérique liant une entité physique à une entité numérique. Une autorité de certification (notée CA *Certification Authority*) fait office de tiers de confiance et atteste le lien. Ainsi, une entité peut ajouter son certificat lors de la signature du message avant de l'envoyer. Ce certificat contiendra entre autre sa clé publique. Le standard le plus utilisé pour la création des certificats numériques est le X.509.

ANNEXE B

AODV

Dans cet annexe, nous présentons les structures maintenues par chaque nœud exécutant AODV. Nous nous intéressons aussi à la structure des messages échangées entre les nœuds lors du processus de création et de maintien des routes.

B.1 Structures maintenues par chaque nœud

B.1.1 Table de routage

La gestion d'une table de routage s'impose puisqu'il s'agit d'un protocole de routage. Les informations sur les routes doivent être conservé même pour les liaisons de courte durées. La structure de cette table est présenté dans la figure B.1.

@D	#SN	Valid_SN	State	Interface	#HC	@NH	PL	LT

FIGURE B.1 – Format de la table de routage

- @D : L'adresse IP de la destination.
- #SN (*Sequence Number of destination*) : Numéro de séquence de la destination.
- Valid_SND : Drapeau indiquant la validité du numéro de séquence.
- State : Drapeau indiquant l'état de l'entrée (par exemple : *Valid*, *Invalid*, *repairable*, *being repaired*).
- Interface : Interface réseau.

– #HC (*Hop Count*) : Nombre de saut nécessaires pour atteindre la destination.
– @NH (*Next Hop*) : Prochain saut en direction de la destination.
– PL (*Precursor List*) : Liste des précurseurs : c'est la liste des voisins auxquels une réponse de route est généré ou transféré.
– LT (*Life Time*) : Temps au delà duquel la route expire ou est effacée.

B.1.2 Table d'historique

Pour diminuer le nombre de messages qui circulent dans le réseau, AODV ne traite qu'une seule fois un message de demande de route. Ainsi, il garde trace des demandes de route déjà traitées en les stockant dans une structure appelée table d'historique (*buffer*). Donc, si un nœud reçoit de nouveau la même demande de route une seconde fois (ou une $n^{ième}$ fois), il la jette.

Chaque entrée de la table d'historique est composé de :

– #ID : Identifiant de la demande de route RREQ.
– @S : Adresse de de source.
– LT : Temps au delà duquel l'entrée sera effacée.

#ID	@S	LT

FIGURE B.2 – Format de la table d'historique

B.2 Structures des messages échangés

B.2.1 Demande de route RREQ (*Route REQuest*)

Ce message est diffusé lorsqu'un nœud détermine qu'il a besoin d'une route vers une destination et ne dispose pas d'une route disponible. C'est le cas lorsque la destination est inconnue ou lorsque une route précédemment valide dans sa table de routage expire ou est marquée invalide. Le nœud créé le paquet présenté dans la figure B.3.

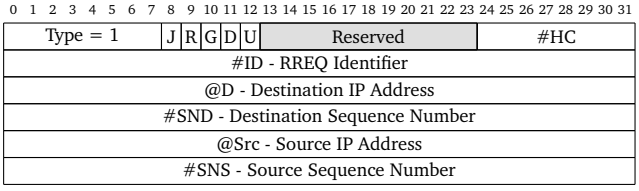


FIGURE B.3 – Format de la demande de route RREQ

– Type : 1.
– J : *Join flag*, réservé au multicast.
– R : *Repair flag*, réservé au multicast.

- G : *Gratuitous RREP flag* : indique la nécessité de générer une réponse de route vers la destination en plus de celle généré vers la source pour l'informer qu'une telle source cherche à la joindre et ainsi un chemin bidirectionnel est établi. Une RREP de ce genre (*gratuitous*) est généré seulement lorsqu'il s'agit d'un nœud intermédiaire qui répond.
- D : *Destination only flag* : indique que seulement la destination doit répondre à cette demande de route.
- U : *Unknown Sequence Number* : indique que le numéro de séquence de la destination est inconnue.
- Reserved : mis à zéro lors de l'envoi et ignorés à la réception.
- #HC (*Hop Count*) : Le nombre de saut de la source de la RREQ au nœud en cours de traitement de la demande.
- #ID : Un numéro de séquence identifiant de manière unique une demande de route lorsqu'il est associé à l'adresse de la source (@Src).
- @D : Adresse IP de la destination à laquelle une route est demandée.
- #SND : Le dernier numéro de séquence connu pour la destination.
- @Src : Adresse IP de la source (nœud qui a initialisé la demande de route).
- #SNS : Numéro de séquence actuel de la source qui sera associé à l'entrée de la table de routage dans les nœuds traitant le message RREQ.

B.2.2 Réponse de route RREP (*Route REPly*)

Lorsqu'une demande de route atteint la destination ou un nœud ayant un chemin valide vers la destination, celui-ci génère une réponse de route qui sera envoyé en unicast d'un nœud à un autre jusqu'à atteindre la source. Le paquet de réponse de route est représenté par la figure B.4.

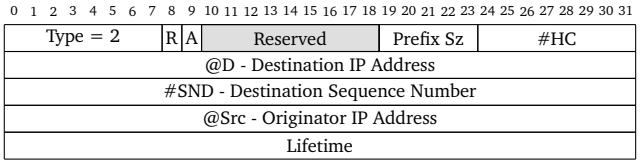


FIGURE B.4 – Format de la réponse de route RREP

- Type : 2.
- R : *Repair flag*, utilisé en multicast.
- A : Accusé de réception requis.
- Reserved : mis à zéro lors de l'envoi et ignorés à la réception.
- Prefix Sz : Si c'est différent de zéro, cela signifie que le prochain saut peut être utilisé pour n'importe quel nœud avec le même préfixe.
- #HC (*Hop Count*) : Nombre de sauts de la destination de la RREQ au nœud en cours de traitement.
- @D : Adresse IP de la destination à laquelle une route est demandée.
- #SND : Numéro de séquence de la destination.
- @Src : Adresse IP de la source, nœud qui a initialisé la demande de route RREQ.

– LifeTime : Temps en milli-secondes pour lequel les nœuds recevant la RREP considèrent la route valide.

B.2.3 HELLO

Les messages HELLO offrent des informations sur la connectivité. Ils sont utilisés par seulement les nœuds faisant partie d'une route active pour valider les connexions avec les voisins. Ainsi, à chaque intervalle (*Hello_Interval*), le nœud vérifie qu'il a diffusé au moins un message et s'il ne l'a pas fait, il envoi une réponse de route avec un TTL (*Time To Live*) égal à 1 : Il s'agit du message HELLO.

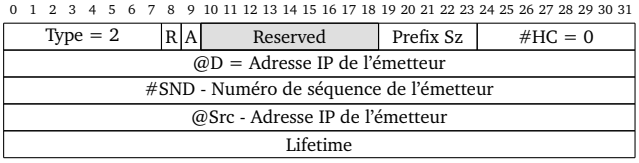


FIGURE B.5 – Format du message HELLO

À chaque fois qu'un nœud reçoit un message (HELLO ou autre) de son voisin X, il remet à zéro le compteur (*delete_period*). Mais, s'il ne reçoit rien et la période est écoulée, le nœud suppose que le lien avec ce voisin X est perdu.

B.2.4 Erreur de route RERR (*Route ERRor*)

Une erreur de route est envoyée à chaque fois que la rupture d'un lien rend inaccessible l'accès à une ou plusieurs destinations.

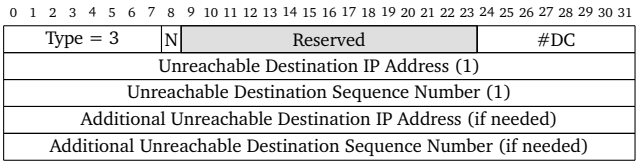


FIGURE B.6 – Format de l'erreur de route RRER

– Type : 3.
– N : *No delete flag* : mis à un lorsque le nœud effectue une réparation locale de la route et les nœuds en amont ne doivent pas effacer la route.
– Reserved : mis à zéro lors de l'envoi et ignorés à la réception.
– #DC : *Destination Count* : Le nombre des destinations non joignables incluses dans le message. Cette valeur doit être au minimum 1.
– *Unreachable Destination IP Address* : L'adresse IP de la destination qui n'est plus accessible.
– *Unreachable Destination Sequence Number* : Le numéro de séquence de la destination (pris de la table de routage) dont l'adresse IP est juste au dessus.

B.2.5 Accusé de réception de réponse de route RREP-ACK

L'accusé de réception (*ACKnowledgment*) doit être envoyé en réponse à une RREP dont le bit "A" est à 1. Ceci est utilisé lorsque le nœud craint l'existence d'un lien unidirectionnel empêchant l'achèvement du processus de découverte de route.

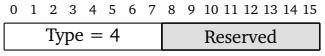


FIGURE B.7 – Format de l'accusé de réponse de route RREP-ACK

- Type : 4.
- Reserved : mis à zéro lors de l'envoi et ignorés à la réception.

ANNEXE C

PSEUDO-CODE DE rreq_process ET rrep_process DE L'IMPLÉMENTATION AODV-UU

Dans ce qui suit, nous présentons sous forme de pseudo-code l'implémentation des deux fonctions `rreq_process` et `rrep_process`. Ces fonctions permettent de mieux comprendre la logique utilisée pour l'implémentation et aussi vérifier la correspondance avec la RFC 3561 d'AODV.

Une étude approfondie de ces fonctions nous a permis de choisir l'emplacement idéal pour insérer notre raisonnement : il s'agit de la partie encadrée.

Algorithme C.1: rreq_process

Entrées : rreq, rreqlen, ip_src, ip_dst, ip_ttl, ifindex

1 **Début**

`// Vérifier la conformité de la longueur du message reçu`

2 **Si** *(rreqlen < RREQ_SIZE)* **Alors**

3 **Return** ;

`// Vérifier si le lien avec l'émetteur n'est pas unidirectionnel`

4 **Si** *(Lien unidirectionnel)* **Alors**

5 **Return** ;

6 **Si** *(rreq déjà traité)* **Alors**

`// À travers la consultation de la table d'historique`

Si *(reasoning_mode == 1)* **Alors**

`// Le nœud M reçoit une RREQ provenant du nœud N`

Si *((RREQ ∈ THE) && (ip_src de la RREQ trouvé dans la THE == N))* **Alors**

N a rejoué une RREQ

7

`// Le nœud M reçoit/entendune RREQ du nœud N`

Si *((RREQ ∈ EHT) and (#SNS received ≠ #SNS found in EHT))* **Alors**

N modified the source sequence number of a RREQ

`// Enrichir la connaissance en stockant les messages normalement ignorés`

Ajouter la RREQ dans la Table d'Historique Étendue THE ;

8 **Return** ;

`// Pour ne pas traiter plusieurs fois la même RREQ`

9 Ajouter rreq à la table d'historique TH (il s'agit de l'ajout de l'identifiant et de l'adresse de la source associés à une durée de vie) ;

`// Cette action est un complément de la ligne 9 où des champs supplémentaires sont stockés pour enrichir la connaissance.`

10 **Si** *(reasoning_mode == 1)* **Alors**

Ajouter la RREQ dans la Table d'Historique Étendue THE ;

`// Toujours ajouter ou mettre à jour le chemin à la source de la RREQ`

11 **Si** *(rreq.@SRC ∈ RT)* **Alors**

12 Mettre à jour le chemin à la source rreq.@SRC dans la table de routage ;

13 **Sinon**

14 Ajouter une entrée à la source rreq.@SRC dans la table de routage ;

⋮

```
    ⋮
    Si (rreq.@D = node_id) Alors
        // Je suis la destination du message, je dois initier une RREP
15      Créer et envoyer une RREP ;
    Sinon
        // Je suis un nœud intermédiaire
        // Je vérifie si j'ai une route active vers la destination
16      Si (∃ route active et valide vers @D dans RT) Alors
17          Si (rreq.destination_only==0 && rreq.#SND < RT.@D.#SND) Alors
                // J'ai un chemin valide et j'ai la possibilité de répondre
18              Créer et envoyer une RREP ;
19          Sinon
                // Je ne peux pas créer une RREP
20              Si (ip_ttl > 1) Alors
21                  Renvoyer la demande de route RREQ ;
22                  Return ;
            // Vérifier l'activation du drapeau de réponse gratuite
23          Si (rreq.gratuitous_flag) Alors
24              Créer et envoyer une RREP gratuite en direction de @D ;
25          Return ;
        // Si aucune condition n'est vérifié, retransmettre la RREQ
26      Si (ip_ttl > 1) Alors
27          Renvoyer la demande de route RREQ ;
```

Algorithme C.2: rrep_process

Entrées : rrep, rreplen, ip_src, ip_dst, ip_ttl, ifindex

```
1  begin
       // Vérifier la conformité de la longueur du message reçu
2      Si (rreplen < RREP_SIZE) Alors
3          Return ;

           // Ignorer les messages qui ne sont pas destinés au nœud traitant
           pour AODV normal (pas de mode raisonnement). Le cas se présente après
4          activation du mode promiscious.
           Si ((reasoning_mode == 0) && (ip_dst ≠ node_id)) Alors
               Return ;

       // Ignorer les messages que j'ai initialisé
5      Si (ip_src == node_id) Alors
6          Return ;

           Si (reasoning_mode == 1) Alors
               // Le nœud M reçoit une RREP du nœud N
               Si ((RREP ∈ THE) && (ip_src de la RREP trouvé dans la THE == N)) Alors
                   N a rejoué une RREP

               // Le nœud M reçoit une RREP du nœud N
               Si ((Pas de RREQ dans la THE correspondant au RREP reçue)
                       && (∄ RREP concernant @D dans la THE avec ip_src==N)) Alors
7                  N a fabriqué une RREP

               // Le nœud M reçoit/entend une RREP du nœud N
               Si ((RREP ∈ THE) && (#SND reçue ≠ #SND trouvé dans la THE)) Alors
                   N a modifié le numéro de séquence de la destination dans la RREP

               // Enrichir la connaissance en stockant les messages de réponses
               de route
               Ajouter la RREP dans la Table d'Historique Étendue THE ;

       // Ignorer les messages qui ne me sont pas destinés
8      Si (ip_dst == node_id) Alors
9          Return ;

       // Vérifier la création d'un chemin vers la destination
10     Si (∄ de chemin vers rrep.@D dans RT) Alors
11         Ajouter un chemin vers rrep.@D
12     Si (rrep.#SND est plus frais ou en cas d'égalité rrep.#HC est plus petit) Alors
13         Mise à jour du chemin vers rrep.@D

       ⋮
```

```
    ⋮
    // Vérifier si un accusé de réception est demandé
14  Si (rrep.acknowlegment) Alors
15      Créer et envoyer l'accusé de réception
    // Vérifier si cette réponse de route m'est destiné
16  Si (ip_dst == node_id) Alors
17      Si (J'ai un chemin vers la rrep.@SRC dans RT et son état est valide) Alors
18          Retransmettre la RREP vers le prochain saut en direction de rrep.@SRC
19      Sinon
20          Créer et envoyer un message d'erreur
```

BIBLIOGRAPHIE

[ABS09] A. Adnane, C. Bidan, and RTD. Sousa. Trust-based countermeasures for securing OLSR protocol. In *Proc. of 2009 International Conference on Computational Science and Engineering (CSE'09)*, volume 2, pages 745–752. IEEE Computer Society, August 2009.

[ACJ+03] C. Adjih, T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, and D. Raffo. Securing the OLSR protocol. In *Proc. of Med-Hoc-Net*, pages 25–27. Citeseer, 2003.

[ASBM07] A. Adnane, RTD Sousa, C. Bidan, and L. Mé. Analysis of the implicit trust within the OLSR protocol. In *Proc. Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'07)*, volume 238, pages 75–90. Springer, July 2007.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems (TOCS)*, 8(1) :18–36, 1990.

[BBK94] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Proc.of European Symposium on Research in Computer Security (ESORICS 94)*, pages 3–18. LNCS 875, Springer-Verlag, 1994.

[BGN87] D.P. Bertsekas, R. Gallager, and T. Nemetz. *Data networks*. Prentice-hall Englewood Cliffs, NJ, 1987.

[BLB02] S. Buchegger and J.Y. Le Boudec. Nodes bearing grudges : towards routing security, fairness, and robustness in mobile ad hoc networks. In *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP 2002)*, pages 403–410. IEEE Computer Society, January 2002.

[BLB05] S. Buchegger and J.Y. Le Boudee. Self-policing mobile ad hoc networks by reputation systems. *IEEE Communications Magazine*, 43(7) :101–107, July 2005.

[Bok99] S. Bok. *Lying : Moral choice in public and private life*. Vintage, 1999.

[BTA+99] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. GloMoSim : A scalable network simulation environment. Technical Report 990027, University od California, Los Angelos, Computer Science Department, May 1999.

[CCBNR07] F. Cuppens, N. Cuppens-Boulahia, S. Nuon, and T. Ramard. Property based intrusion detection to secure OLSR. In *Proc. 3rd IEEE International Conference on Wireless and Mobile Communications (ICWMC'07)*, pages 52–60. IEEE Computer Society, March 2007.

[CDD10] T. Clausen, C. Dearlove, and J. Dean. Mobile ad hoc network (manet) neighborhood discovery protocol (nhdp). http ://tools.ietf.org/html/draft-ietf-manet-nhdp-12, March 2010. Internet-Draft.

[Cha99] X. Chang. Network simulations with OPNET. In *Proc. of the 31st conference on Winter simulation : Simulation - a bridge to the future*, volume 1, pages 307–314. ACM, 1999.

[CJ03] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol OLSR. http ://tools.ietf.org/html/rfc3626, October 2003. RFC3626.

[CKS+06] T. Condie, V. Kacholia, S. Sankararaman, J. Hellerstein, and P. Maniatis. Induced churn as shelter from routing-table poisoning. In *Proc. of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, 2006.

[CM99] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET) : Routing Protocol Performance Issues and Evaluation ConsidÃ©rations. http ://tools.ietf.org/html/rfc2501, January 1999. Informational Memo.

[Deu62] M. Deutsch. *Cooperation and Trust : Some Theoretical Notes*. Nebraska Symposium on Motivation. Nebraska University Press, 1962.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1) :269–271, 1959.

[DLA02] H. Deng, W. Li, and DP Agrawal. Routing security in wireless ad hoc networks. *IEEE Communications Magazine*, 40(10) :70–75, 2002.

[Dou02] J. Douceur. The sybil attack. *Peer-to-Peer Systems*, 1 :251–260, 2002.

[DR02] J. Daemen and V. Rijmen. *The design of Rijndael : AES–the advanced encryption standard*. Springer, 2002.

[Fri07] M. Frikha. *Planification et simulation des réseaux*. Collection performance des réseaux. AndrÃ©-Luc Beylot, 1 edition, September 2007.

[GAB06] L. Guang, C. Assi, and A. Benslimane. Interlayer Attacks in Mobile Ad hoc Networks. In *Proc. of the Second International Conference on Mobile ad-hoc and sensor networks (MSN 2006)*, pages 436–448. Springer, December 2006.

[Gam00] D. Gambetta. *Can we trust trust*, chapter 13, pages 213–237. Trust : Making and Breaking Cooperative Relations, electronic edition, Department of Sociology, University of Oxford, 2000.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proc. of IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, May 1990.

[GS03] S. Gupte and M. Singhal. Secure routing in mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1) :151–174, 2003.

[Haa97] ZJ Haas. A new routing protocol for the reconfigurable wireless networks. In *Proc.of IEEE 6th International Conference on Universal Personal Communications Record.*, volume 2, pages 562–566, 1997.

[HJP03] Y.C. Hu, D.B. Johnson, and A. Perrig. SEAD : secure efficient distance vector routing for mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1) :175–192, July 2003.

[HL04] Y. Huang and W. Lee. Attack Analysis and Detection for Ad Hoc Routing Protocols. In *Proc. of 7th International Symposium on Recent Advances in Intrusion Detection RAID*, volume 3224, pages 125–145. Springer, September 2004.

[HPJ03] Y.C. Hu, A. Perrig, and D.B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *Proc. of the 2nd ACM workshop on Wireless security*, page 40. ACM, 2003.

[HPJ05] Y.C. Hu, A. Perrig, and D.B. Johnson. Ariadne : A Secure On-Demand Routing Protocol for Ad Hoc Networks. *Wireless Networks (WiNet)*, 11 :21–38, January 2005.

[HPJ06] Y.C. Hu, A. Perrig, and DB Johnson. Wormhole attacks in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2) :370–380, 2006.

[HPS02a] Z.J. Haas, M.R. Pearlman, and P. Samar. The bordercast resolution protocol (brp) for ad hoc networks. http ://tools.ietf.org/html/draft-ietf-manet-zone-brp-02, July 2002. Internet-Draft.

[HPS02b] Z.J. Haas, M.R. Pearlman, and P. Samar. The interzone routing protocol (ierp) for ad hoc networks. http ://tools.ietf.org/html/draft-ietf-manet-zone-ierp-02, July 2002. Internet-Draft.

[HPS02c] Z.J. Haas, M.R. Pearlman, and P. Samar. The intrazone routing protocol (iarp) for ad hoc networks. http ://tools.ietf.org/html/draft-ietf-manet-zone-iarp-02, July 2002. Internet-Draft.

[HPS02d] Z.J. Haas, M.R. Pearlman, and P. Samar. The zone routing protocol (zrp) for ad hoc networks. http ://tools.ietf.org/id/draft-ietf-manet-zone-zrp-04.txt, July 2002. Internet-Draft.

[IH08] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer, December 2008.

[ITS91] ITSEC. *Information Technology Security Evaluation Criteria (ITSEC) : Preliminary Harmonised Criteria. Document COM(90) 314, Version 1.2*. Commission of the European Communities, June 1991.

[JHM07] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. http ://tools.ietf.org/html/rfc4728, February 2007. RFC4728.

[JM96] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. In Thomasz Imielinski and Hank Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[JN99] Mario Joa-Ng. *Routing Protocol and Medium Access Protocol for Mobile Ad Hoc Networks*. PhD thesis, Polytechnic University, NY, USA, January 1999.

[Kah03] A. Kahate. *Cryptography and network security*. Tata McGraw-Hill, 2 edition, 2003.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC : Keyed-Hashing for Message Authentication. http ://www.ietf.org/rfc/rfc2104.txt, February 1997. Internet Draft.

[Ker83] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX :5–83, 1883.

[KW03] C. Karlof and D. Wagner. Secure routing in wireless sensor networks : Attacks and countermeasures. *Ad hoc networks*, 1(2-3) :293–315, 2003.

[LLL04] X. Li, MR Lyu, and J. Liu. A Trust Model Based Routing Protocol for Secure Ad-hoc Networks. In *Proc. Aerospace Conference, (AC'04)*, volume 2, pages 1286–1295. IEEE, December 2004.

[Luh79] N. Luhmann. *Trust and power*. John Willey & Sons, 1979.

[Mar99] S.P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Departement of Computing Scince, Univercity of Stirling, UK, June 1999.

[MC02] G. Montenegro and C. Castelluccia. Statistically unique and cryptographically verifiable (SUCV) identifiers and addresses. In *Proc. of the 9th Annual Network and Distributed System Security Symposium (NDSS'02)*, February 2002.

[MGLB00] S. Marti, TJ Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 255–265. ACM, August 2000.

[MM02] P. Michiardi and R. Molva. CORE : A Collaborative Reputation Mechanism to Enforce Node Cooperation in Mobile Ad hoc Networks. In *Proc. IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security : Advanced Communications and Multimedia Security (CMS'02)*, volume 228, pages 107–121. Kluwer, September 2002.

[MMH02] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. In *Proc. of the 35th Annual Hawaii International Conference on System Sciences (HICSS)*, pages 2431–2439, 2002.

[MRM04] C. Murthy, Siva Ram, and B.S. Manoj. *Ad Hoc Wireless Networks : Architectures and Protocols*. Prentice Hall PTR, 2004.

[Nor07] Erik Nordstrom. Aodv-uu. http ://core.it.uu.se/core/index.php/AODV-UU, December 2007. Uppsala University.

[NS03] P. Ning and K. Sun. How to misuse AODV : a case study of insider attacks against mobile ad-hoc routing protocols. In *Proc. IEEE Systems, Man and Cybernetics Society, Information Assurance Workshop (IAW'03)*, pages 60–67. IEEE, June 2003.

[OKRK03] P. Obreiter, B. Konig-Ries, and M. Klein. Stimulating cooperative behavior of autonomous devices - an analysis of requirements and existing approaches. In *Proc. of 2nd International Workshop on Wireless Information Systems (WIS2003)*, pages 71–82, 2003.

[OR01] G. O'shea and M. Roe. Child-proof authentication for MIPv6 (CAM). *ACM SIGCOMM Computer Communication Review*, 31(2) :4–8, 2001.

[PA04] N. Prigent and J-P Andreaux. Gestion sécurisée de groupes de dispositifs dans les réseaux domestiques. In *Symposium sur la Sécurité des Technologies de l'Information et de la Communication (SSTIC'04)*, Juin 2004.

[PB94] C.E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proc. of the conference on Communications architectures, protocols and applications (SIGCOMM'94)*, pages 244–254. ACM, August 1994.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. http ://tools.ietf.org/html/rfc3561, July 2003. RFC3561.

[PCTS02] A. Perrig, R. Canetti, JD Tygar, and D. Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5(2) :2–13, 2002.

[PJ03] J-M. Percher and B. Jouga. DÃ©tection d'intrusions dans les rÃ©seaux ad hoc. In *Proc. of 1er Symposium sur la SÃ©curitÃ© des Technologies de l'Information et de la Communication (SSTIC'03)*, juin 2003.

[PR99] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings (WMCSA'99)*, volume 2, pages 90–100. IEEE Computer Society, February 1999.

[PW02] K. Paul and D. Westhoff. Context aware detection of selfish nodes in dsr based ad-hoc networks. In *Proc. of IEEE GLOBECOM*, pages 178–182, November 2002.

[RACM04] D. Raffo, C. Adjih, T. Clausen, and P. Muhlethaler. An advanced signature system for OLSR. In *Proc. of the 2nd ACM Workshop on Security of Ad hoc and Sensor Networks*, pages 10–16. ACM, 2004.

[Ran92] P.V. Rangan. An axiomatic theory of trust in secure communication protocols. *Computers & Security*, 11(2) :163–172, 1992.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :126, 1978.

[SA99] F. Stajano and R. Anderson. The resurrecting duckling : Security issues in ad-hoc wireless networks. In *Proc. of the 7th International Workshop on Security Protocols*, pages 172–194. Springer, 1999.

[SAT05] I. Stamouli, P.G. Argyroudis, and H. Tewari. Real-time intrusion detection for ad hoc networks. In *Proc. of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM'05)*, pages 374–380. IEEE Computer Society, 2005.

[SDL+02] K. Sanzgiri, B. Dahill, B.N. Levine, C. Shields, and E.M. Belding-Royer. A Secure Routing Protocol for Ad Hoc Networks. In *Proc. 10th IEEE International Conference on Network Protocols (ICNP'02)*, pages 78–89. IEEE Computer Society, November 2002.

[Shi07] R. Shirey. Internet security glossary, version 2. http ://tools.ietf.org/html/rfc4949, August 2007.

[TBK+03] C.Y. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt. A specification-based intrusion detection system for AODV. In *Proc. 1st ACM workshop on Security of ad hoc and sensor networks (SASN'03)*, pages 125–134. ACM, October 2003.

[VGS+04] G. Vigna, S. Gwalani, K. Srinivasan, EM Belding-Royer, and RA Kemmerer. An intrusion detection tool for AODV-based ad hoc wireless networks. In *Proc. of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 16–27. IEEE Computer Society, 2004.

[WLMG05] M. Wang, L. Lamont, P. Mason, and M. Gorlatova. An Effective Intrusion Detection Approach for OLSR MANET Protocol. In *Proc. 1st Workshop on Secure Network Protocols (NPSec'05)*, pages 55–60. IEEE, November 2005.

[YKB93] R. Yahalom, B. Klein, and T. Beth. Trust Relationships in Secure Systems-A Distributed Authentication Perspective. In *Proc. IEEE Symposium on Research in Security and Privacy (SP'93)*, pages 150–164. IEEE Computer Society, May 1993.

[ZA02] M.G. Zapata and N. Asokan. Securing ad hoc routing protocols. In *Proc. 1st ACM workshop on Wireless security (WiSE'02)*, pages 1–10. ACM, September 2002.

VU :
Le Directeur de Thèse

VU :
Le Responsable de l'école Doctorale

VU pour autorisation de soutenance

Rennes, le

Le Président de l'Université de Rennes 1

Monsieur Guy Cathelineau

VU après soutenance pour autorisation de publication :
Le Président du Jury,

Résumé

La constante évolution des technologies de l'information et le penchant vers l'utilisation des machines sans fil qui se sont imposées ces dernières années ont fait émerger un nouveau type de réseaux : les réseaux sans-fil ad hoc, ou MANET (*Mobile Ad hoc NETwork*). Se souciant de pouvoir communiquer et de partager l'information dans n'importe quelle situation, les réseaux ad hoc sont des systèmes autonomes composés par un ensemble d'entités mobiles libres de se déplacer sans contraintes. Ces entités utilisent le médium radio pour communiquer et forment un réseau n'utilisant aucune infrastructure existante.

Dans cette thèse, nous nous intéressons à sécuriser les protocoles de routage ad hoc en proposant des mécanismes de détection des actions malveillantes pour consolider les protocoles de routage et prévenir les futures attaques. Nous nous basons sur la confiance développée entre les entités pour bâtir un raisonnement sur le comportement des entités voisines. Ceci passe par une comparaison des messages échangés entre les entités du réseau ad hoc ou encore par l'analyse des incohérences dans les annonces et les ouvertures de routes afin de détecter la malhonnêteté ou la compromission d'une entité mobile. Nous choisissons d'appliquer cette approche sur le protocole réactif AODV.

L'évaluation des performances de ce système de détection montre la pertinence du raisonnement sans pour autant influencer les performances du protocole. Une étude portant sur les cas de faux-positifs est aussi effectuée : nous proposons une solution pour les limiter en se basant sur le fait de ne prendre une décision que si l'entité est sûre de l'action malhonnête du voisin.

Abstract

An ad hoc network is a collection of mobile devices that communicate in a self organized way using wireless network interfaces without neither centralized administration nor fixed infrastructure. In such a network, nodes must cooperate with each other so as to extend their transmission range to reach distant nodes. This cooperation requires a specific ad hoc routing protocol to establish and maintain routes between nodes. Ad hoc routing protocols are based on mutual trust between collaborating nodes and suppose a correct behavior. However, as some nodes that legitimately belong to the network may lie so as to manipulate it to their advantage, every node should consider its environment as hostile and implement accordingly its own security mechanisms to protect itself against internal (i.e. a priori trusted but in fact not trustworthy) dishonest nodes.

In this thesis, we are interested in detecting misbehaving nodes within the ad hoc routing protocol AODV. We propose and implement a detection system based on implicit trust relations : a node implementing this system collects its neighbors' routing messages and reasons about them to decide on their trustworthiness. We also evaluate our implementation, and, based on simulations, show that the system we have developed to detect dishonest behavior is efficient. We also study cases of false positive and we propose a solution to limit their occurrences.